US007542980B2

(12) **United States Patent**
Tsyganskiy et al.

(10) **Patent No.: US 7,542,980 B2**
(45) **Date of Patent: Jun. 2, 2009**

(54) **METHODS OF COMPARING AND MERGING BUSINESS PROCESS CONFIGURATIONS**

(75) Inventors: **Igor Tsyganskiy**, Palo Alto, CA (US); **Vitaliy Stulski**, San Mateo, CA (US); **Dmitri Liakh**, Foster City, CA (US); **Eugene Satsuta**, Saratoga, CA (US); **Luiz Scheinkman**, Sunnyvale, CA (US)

(73) Assignee: **SAP AG**, Walldorf (DE)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 587 days.

(21) Appl. No.: **11/203,262**

(22) Filed: **Aug. 15, 2005**

(65) **Prior Publication Data**

US 2006/0242207 A1 Oct. 26, 2006

Related U.S. Application Data

(63) Continuation-in-part of application No. 11/111,794, filed on Apr. 22, 2005.

(51) **Int. Cl.**
G06F 17/00 (2006.01)

(52) **U.S. Cl.** **707/101**; 707/100; 707/102; 707/104.1

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,499,371 A 3/1996 Henninger et al.
5,542,078 A 7/1996 Martel et al.
5,596,746 A 1/1997 Shen et al.
5,659,723 A 8/1997 Dimitrios et al.
5,812,850 A 9/1998 Wimble
5,917,730 A 6/1999 Rittie et al.
5,983,020 A 11/1999 Sweeney et al.
6,023,578 A 2/2000 Birsan et al.
6,031,993 A 2/2000 Andrews et al.
6,115,646 A 9/2000 Fiszman et al.
6,128,771 A 10/2000 Tock et al.
6,163,781 A 12/2000 Wess, Jr.
6,269,475 B1 7/2001 Farrell et al.
6,298,354 B1 10/2001 Saulpaugh et al.
6,360,223 B1 3/2002 Ng et al.

(Continued)

OTHER PUBLICATIONS

"Eclipse Platform Technical Overview," IBM Corp. and the Eclipse Foundation, Dec. 2005 (23 pages).

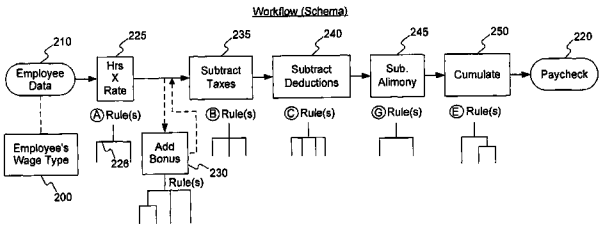
(Continued)

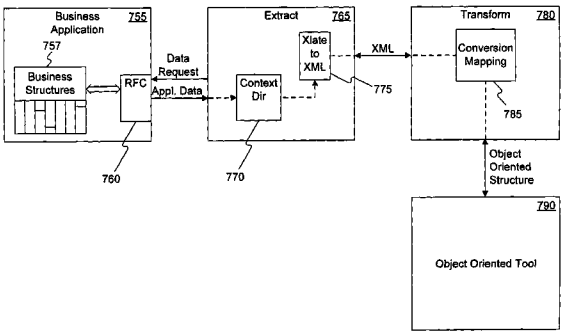
Primary Examiner—Sana Al-Hashemi
(74) *Attorney, Agent, or Firm*—Finnegan, Henderson, Farabow, Garrett & Dunner, LLP

(57) **ABSTRACT**

Exemplary methods and systems consistent with the present invention allow a user to compare different versions of a business application using an object-oriented model of the business application. By transforming business application structures into object-oriented structures, such as object-oriented language constructs, structures, and other elements, individual differences between versions of the business applications may be identified and displayed to a user, for example, on a user interface. A user may display object-oriented models of the business application representing, for instance, the latest control version, the latest deployed version, and a test version including test modifications. Having all three versions represented in object-oriented models allows a user to perform a three-way comparison between the object-oriented models, increasing productivity and efficiency of maintaining, debugging, modifying, controlling and deploying the business application.

20 Claims, 61 Drawing Sheets





U.S. PATENT DOCUMENTS

6,374,252	B1	4/2002	Althoff et al.	
6,374,256	B1	4/2002	Ng et al.	
6,467,079	B1	10/2002	Ettritch et al.	
6,502,098	B2	12/2002	Lau	
6,542,937	B1	4/2003	Kask et al.	
6,625,651	B1	9/2003	Swartz et al.	
6,643,664	B1	11/2003	Yaung	
6,668,370	B1	12/2003	Harmon et al.	
6,681,383	B1	1/2004	Pastor et al.	
6,754,670	B1	6/2004	Lindsay et al.	
6,772,172	B2	8/2004	Leong et al.	
6,862,711	B1	3/2005	Bahrs et al.	
7,003,781	B1	2/2006	Blackwell et al.	
7,047,524	B1	5/2006	Braddock, Jr.	
7,072,913	B2	7/2006	Jans et al.	
7,072,934	B2	7/2006	Helgeson et al.	
7,099,887	B2	8/2006	Hoth et al.	
7,120,896	B2	10/2006	Budhiraja et al.	
7,194,485	B2	3/2007	Kaipa et al.	
7,231,400	B2	6/2007	Cameron et al.	
2002/0083168	A1	6/2002	Sweeney et al.	
2003/0034989	A1	2/2003	Kondo	
2003/0084127	A1	5/2003	Budhiraja et al.	
2003/0110446	A1	6/2003	Nemer	
2004/0002982	A1	1/2004	Ersek et al.	
2004/0006506	A1	1/2004	Hoang	
2004/0025171	A1	2/2004	Barinov et al.	
2004/0030716	A1	2/2004	Hoth et al.	
2004/0111702	A1	6/2004	Chan	
2004/0168124	A1	8/2004	Beisiegel et al.	
2004/0215642	A1	10/2004	Cameron et al.	
2005/0027732	A1	2/2005	Kalima	
2005/0050477	A1	3/2005	Robertson et al.	
2005/0076328	A1	4/2005	Berenbach et al.	
2005/0165822	A1	7/2005	Yeung et al.	
2005/0234916	A1	10/2005	Bergman et al.	
2005/0234976	A1	10/2005	Oara et al.	
2005/0262107	A1 *	11/2005	Bergstraesser et al.	..... 707/100
2005/0278358	A1	12/2005	Doughan	
2006/0010106	A1	1/2006	Gerea et al.	
2006/0106626	A1	5/2006	Jeng et al.	
2006/0149790	A1	7/2006	Rusch	
2006/0241961	A1	10/2006	Tsyganskiy et al.	
2006/0241999	A1	10/2006	Tsyganskiy et al.	
2006/0242170	A1	10/2006	Tsyganskiy et al.	
2006/0242171	A1	10/2006	Tsyganskiy et al.	
2006/0242172	A1	10/2006	Tsyganskiy et al.	
2006/0242173	A1	10/2006	Tsyganskiy et al.	
2006/0242174	A1	10/2006	Tsyganskiy et al.	
2006/0242175	A1	10/2006	Tsyganskiy et al.	
2006/0242176	A1	10/2006	Tsyganskiy et al.	
2006/0242177	A1	10/2006	Tsyganskiy et al.	
2006/0242188	A1	10/2006	Tsyganskiy et al.	
2006/0242196	A1	10/2006	Tsyganskiy et al.	
2006/0242197	A1	10/2006	Tsyganskiy et al.	
2006/0242207	A1	10/2006	Tsyganskiy et al.	
2006/0271911	A1	11/2006	Palapudi et al.	
2006/0282458	A1	12/2006	Tsyganskiy et al.	
2006/0293934	A1	12/2006	Tsyganskiy et al.	
2006/0293935	A1	12/2006	Tsyganskiy et al.	
2006/0293940	A1	12/2006	Tsyganskiy et al.	
2006/0294158	A1	12/2006	Tsyganskiy et al.	
2007/0214173	A1	9/2007	Ohashi et al.	

OTHER PUBLICATIONS

Sharewareconnection, "Software Listing: Schema Compare," Apr. 26, 2008 http://www.sharewareconnection.com/titles/schema-compare.htm (8 pgs).

SQLDBCompare, "The powerful database schema comparison tool for SQL Server databases!," Best SoftTool, Apr. 1, 2004, Apr. 26, 2008 http://web.archive.org/web/20040401202050/http://www.bestsofttool.com/SQLDBCompare/SDC_Introduction.aspx (5 pgs).

Stulski, Vitaliy, "Diagnosing Application Failures in WebLogic," WebLogic Developer's Journal, Sep. 10, 2003, Apr. 29, 2008, http://weblogic.sys-con.com/read/42977.htm> (10 pgs).

U.S. Appl. No. 11/203,273, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,225, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,224, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,275, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,295, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,265, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,226, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,308, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,276, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,292, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,274, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,227, filed Aug. 15, 2005.

U.S. Appl. No. 11/203,293, filed Aug. 15, 2005.

Fahrner, Christian, " A survey of database design transformations based on the Entity Relationship model," Data & Knowledge Engineering 15 (1995) pp. 213-250.

Sinha, Atish P. et al., "An empirical investigation of entity-based and object-oriented data modeling" a development life cycle approach," ACM, Jan. 1999, pp. 229-244.

Tichelaar, Sander, "Modeling Object Oriented Software for Reverse Engineering and Refactoring," Inauguraldissertation de Philosophisch-naturwissenschaftlichen fakultat der Universitat Bern; Dec. 2001 (198 pages).

* cited by examiner

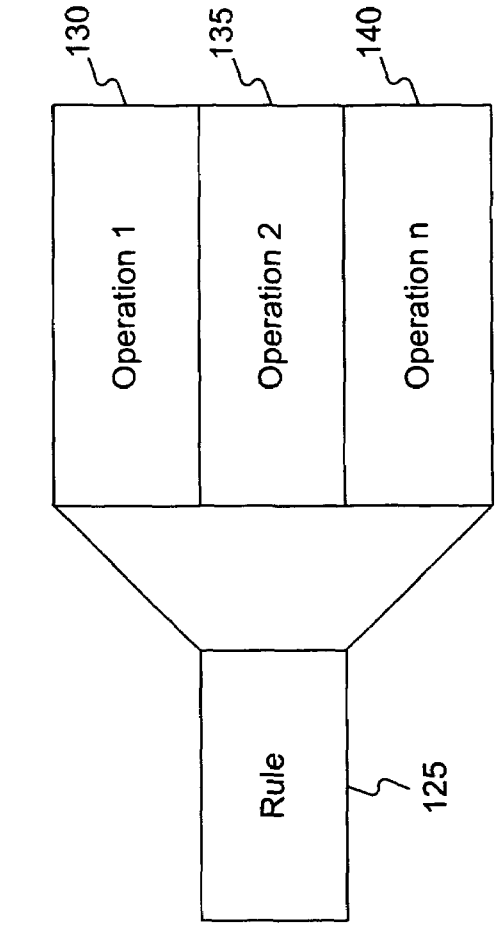


FIGURE 1B

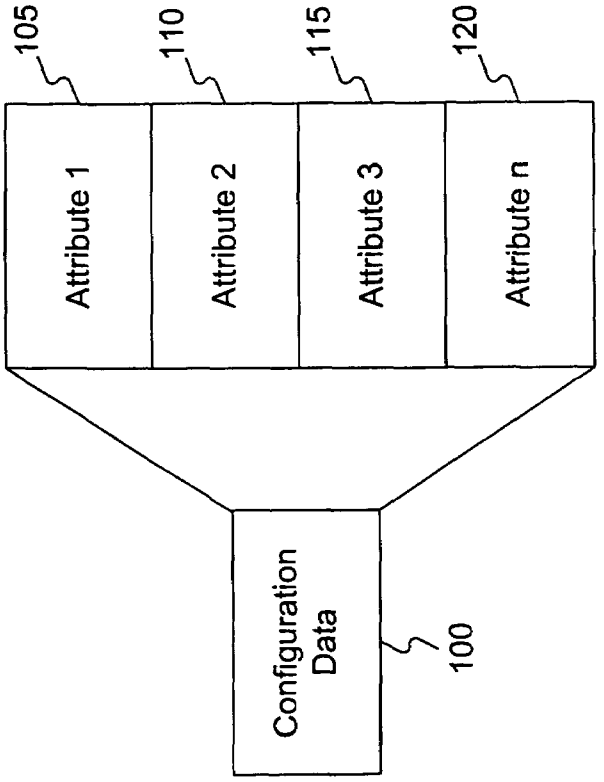


FIGURE 1A

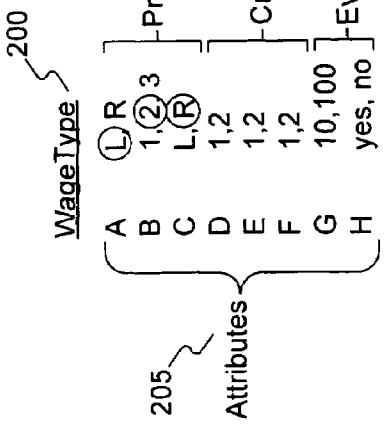


FIGURE 2A

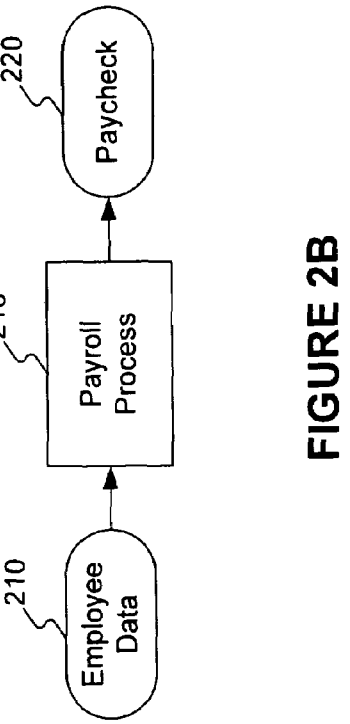


FIGURE 2B

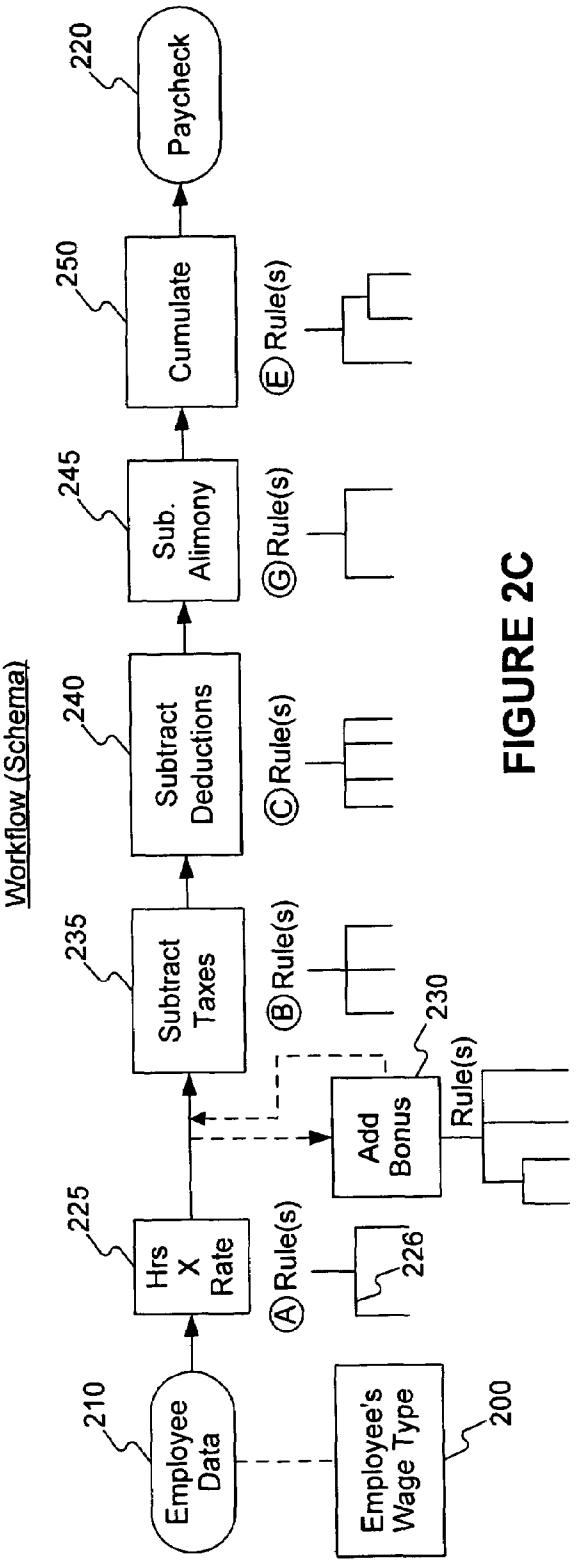


FIGURE 2C

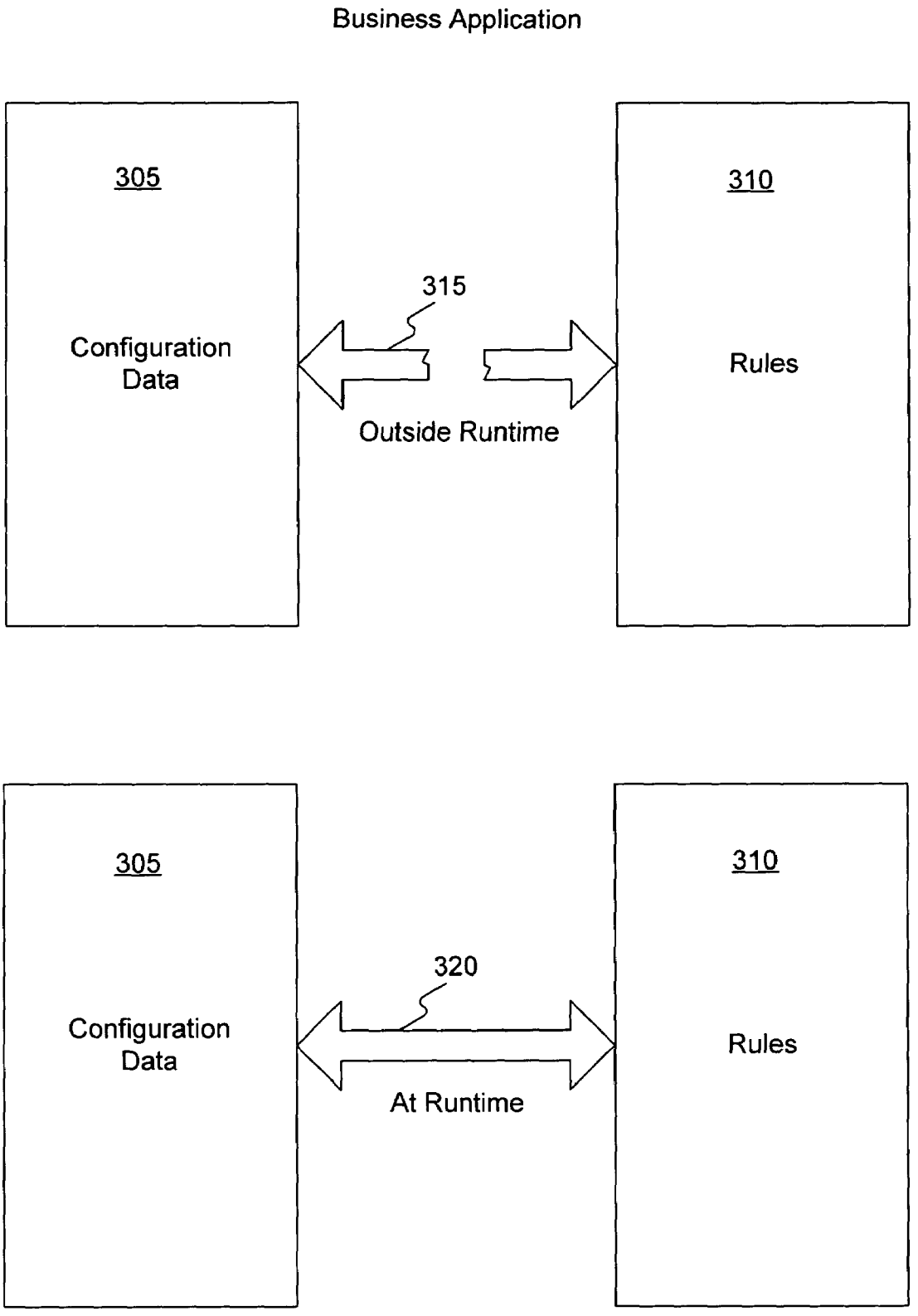

Business Application
305
Configuration Data
315
Outside Runtime
310
Rules
305
Configuration Data
320
At Runtime
310
Rules

FIGURE 3A

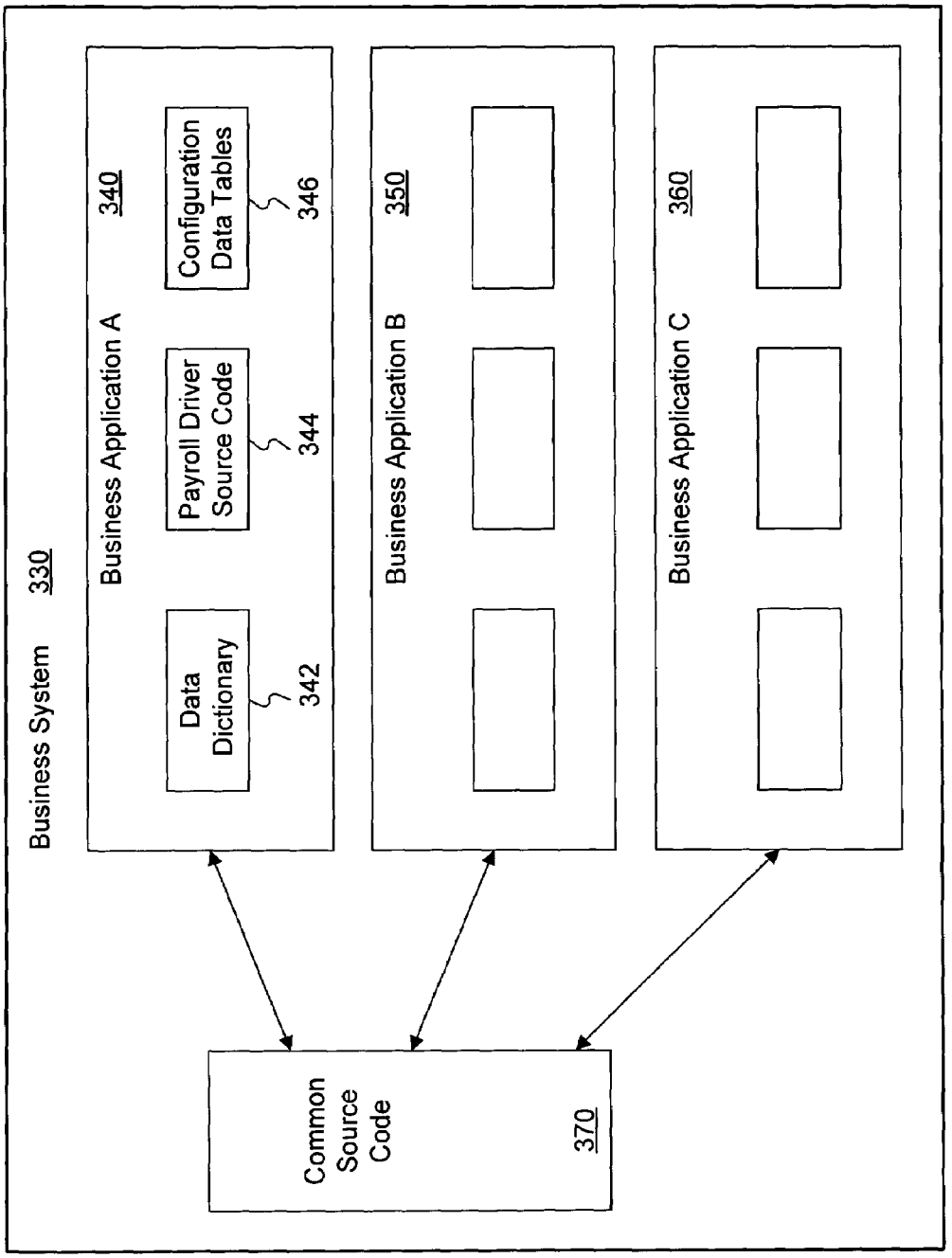

Business System 330
Business Application A 340
Data Dictionary 342
Payroll Driver Source Code 344
Configuration Data Tables 346
Business Application B 350
Business Application C 360
Common Source Code 370

FIGURE 3B

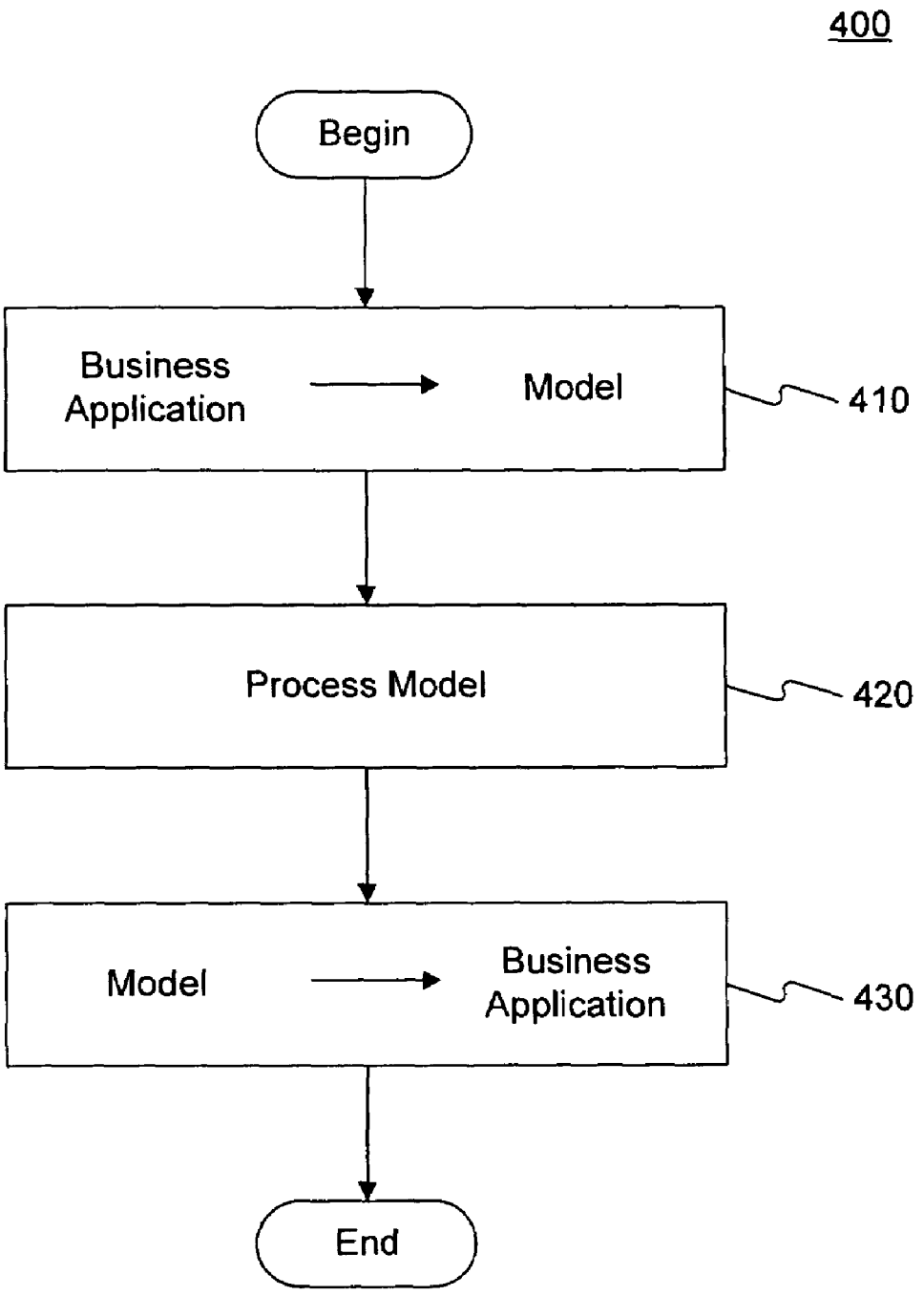

400
Begin
Business Application
Model
410
Process Model
420
Model
Business Application
430
End

FIGURE 4A

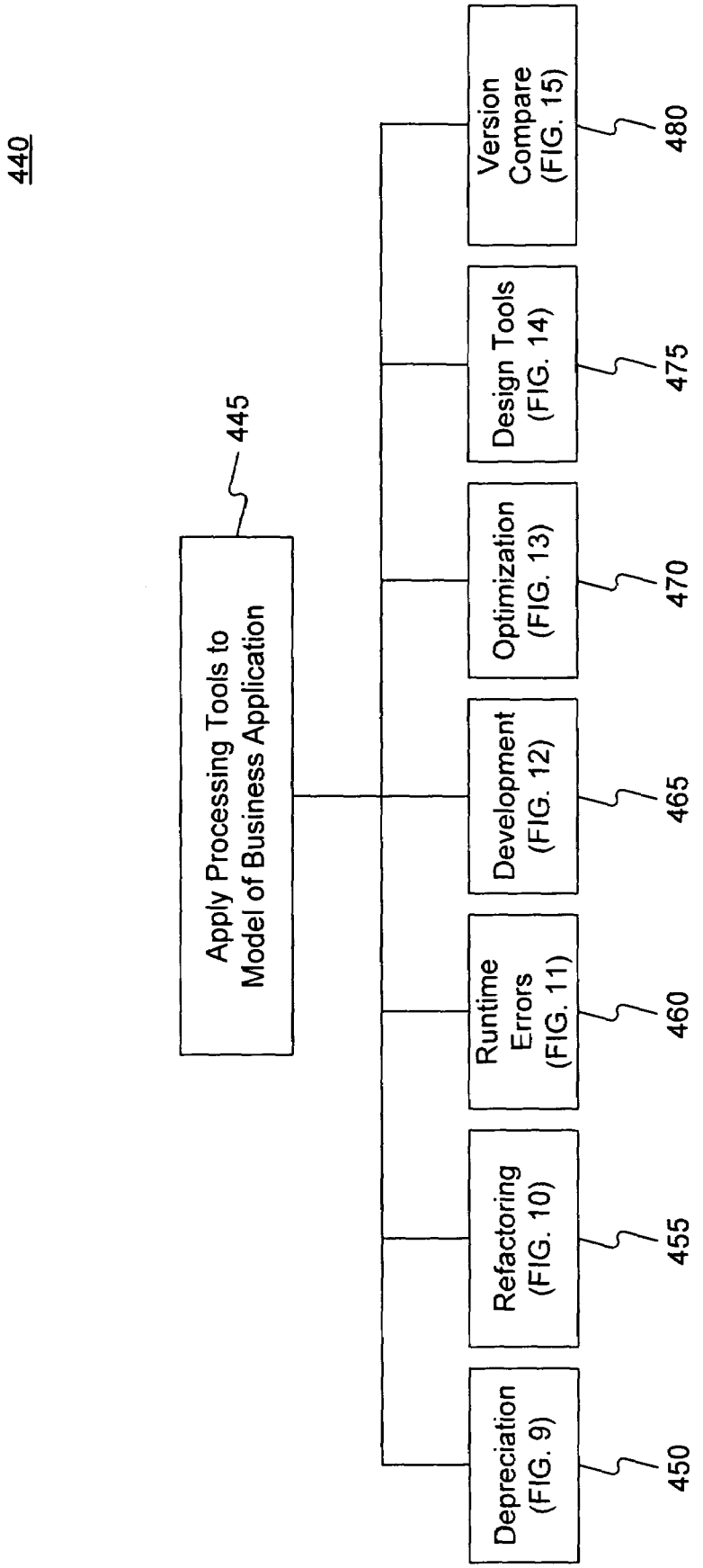

440
Apply Processing Tools to Model of Business Application
445
Depreciation (FIG. 9)
450
Refactoring (FIG. 10)
455
Runtime Errors (FIG. 11)
460
Development (FIG. 12)
465
Optimization (FIG. 13)
470
Design Tools (FIG. 14)
475
Version Compare (FIG. 15)
480

FIGURE 4B

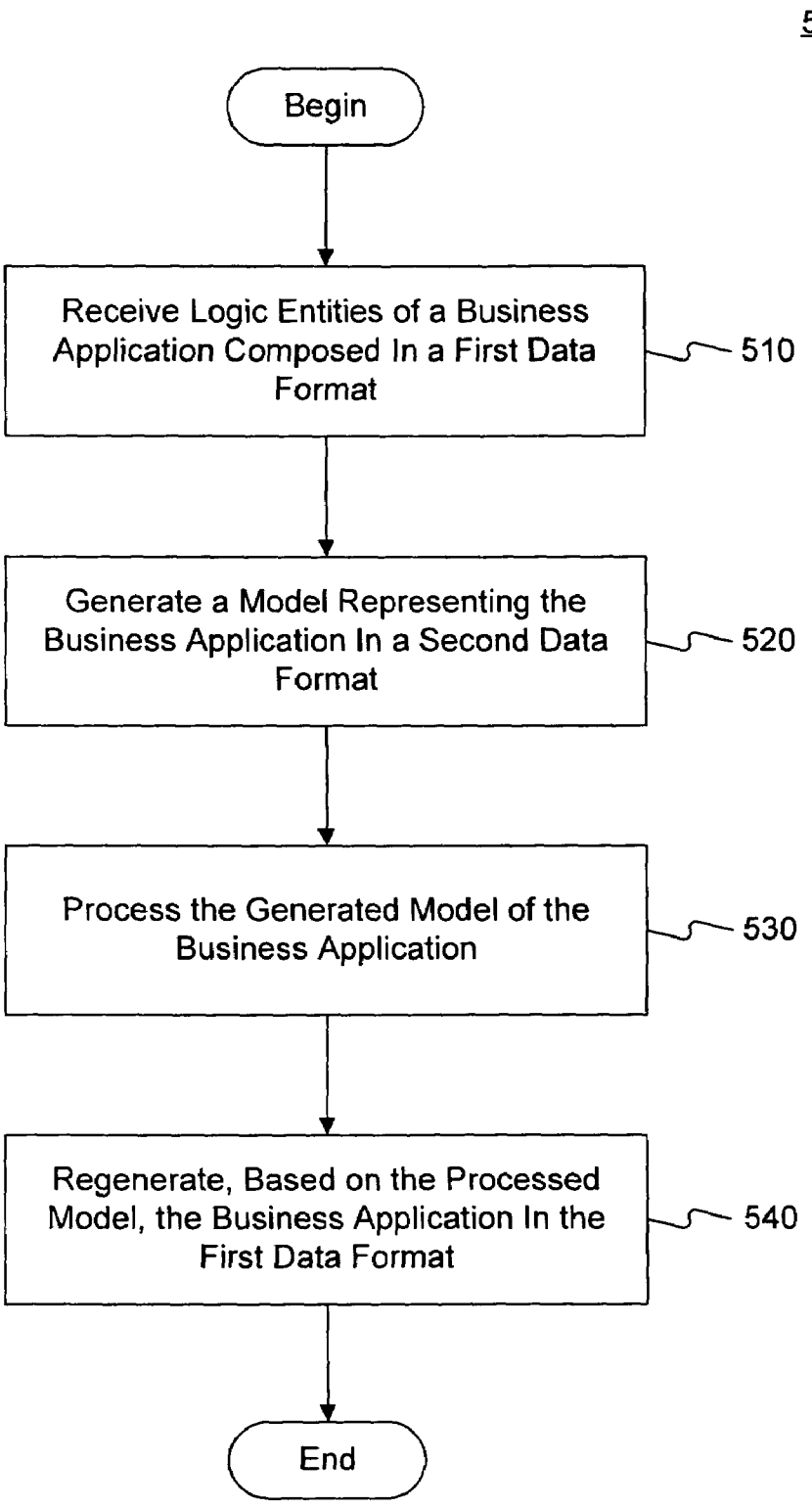

500
Begin
Receive Logic Entities of a Business Application Composed In a First Data Format
510
Generate a Model Representing the Business Application In a Second Data Format
520
Process the Generated Model of the Business Application
530
Regenerate, Based on the Processed Model, the Business Application In the First Data Format
540
End

FIGURE 5

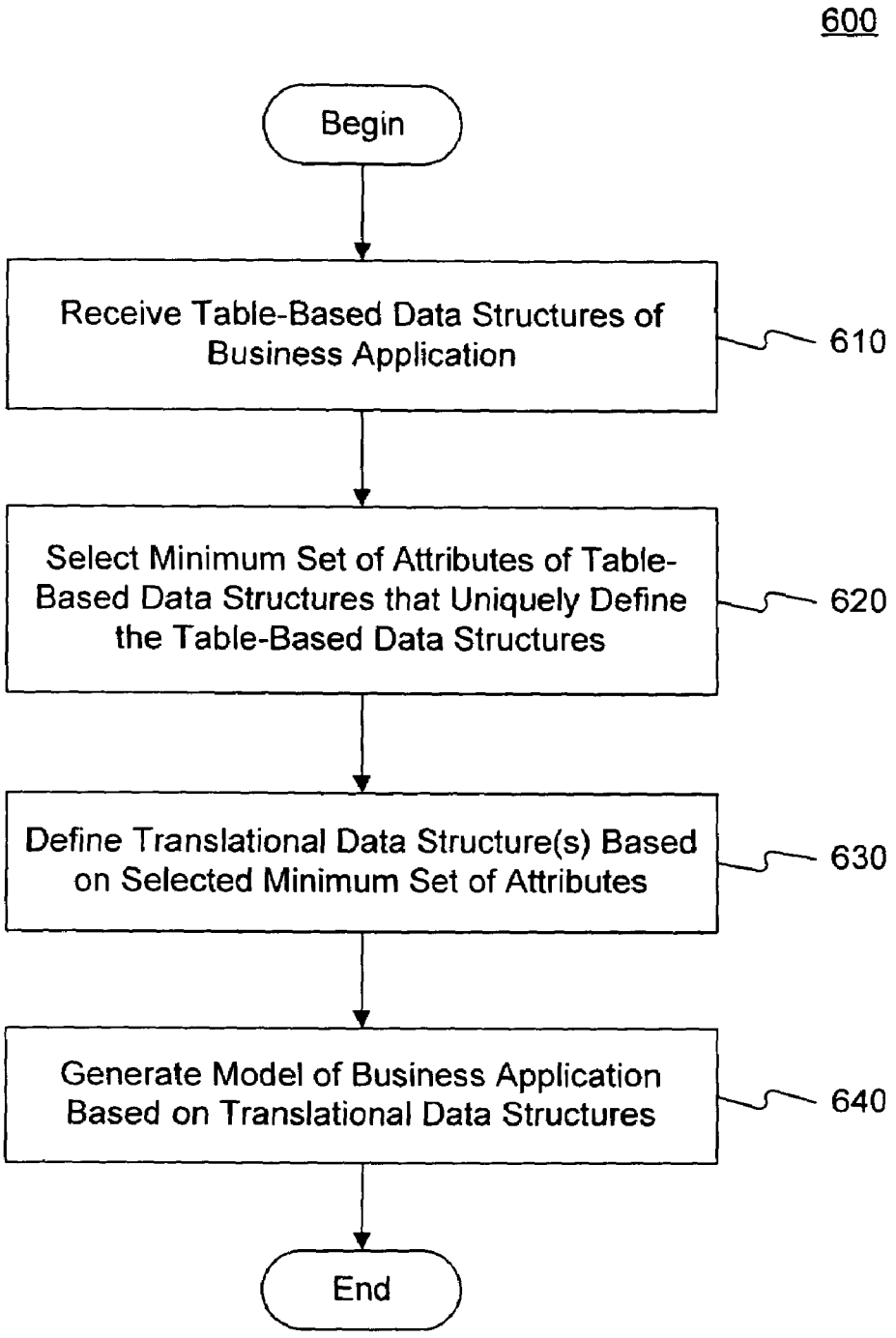

600
Begin
Receive Table-Based Data Structures of Business Application
610
Select Minimum Set of Attributes of Table-Based Data Structures that Uniquely Define the Table-Based Data Structures
620
Define Translational Data Structure(s) Based on Selected Minimum Set of Attributes
630
Generate Model of Business Application Based on Translational Data Structures
640
End

FIGURE 6A

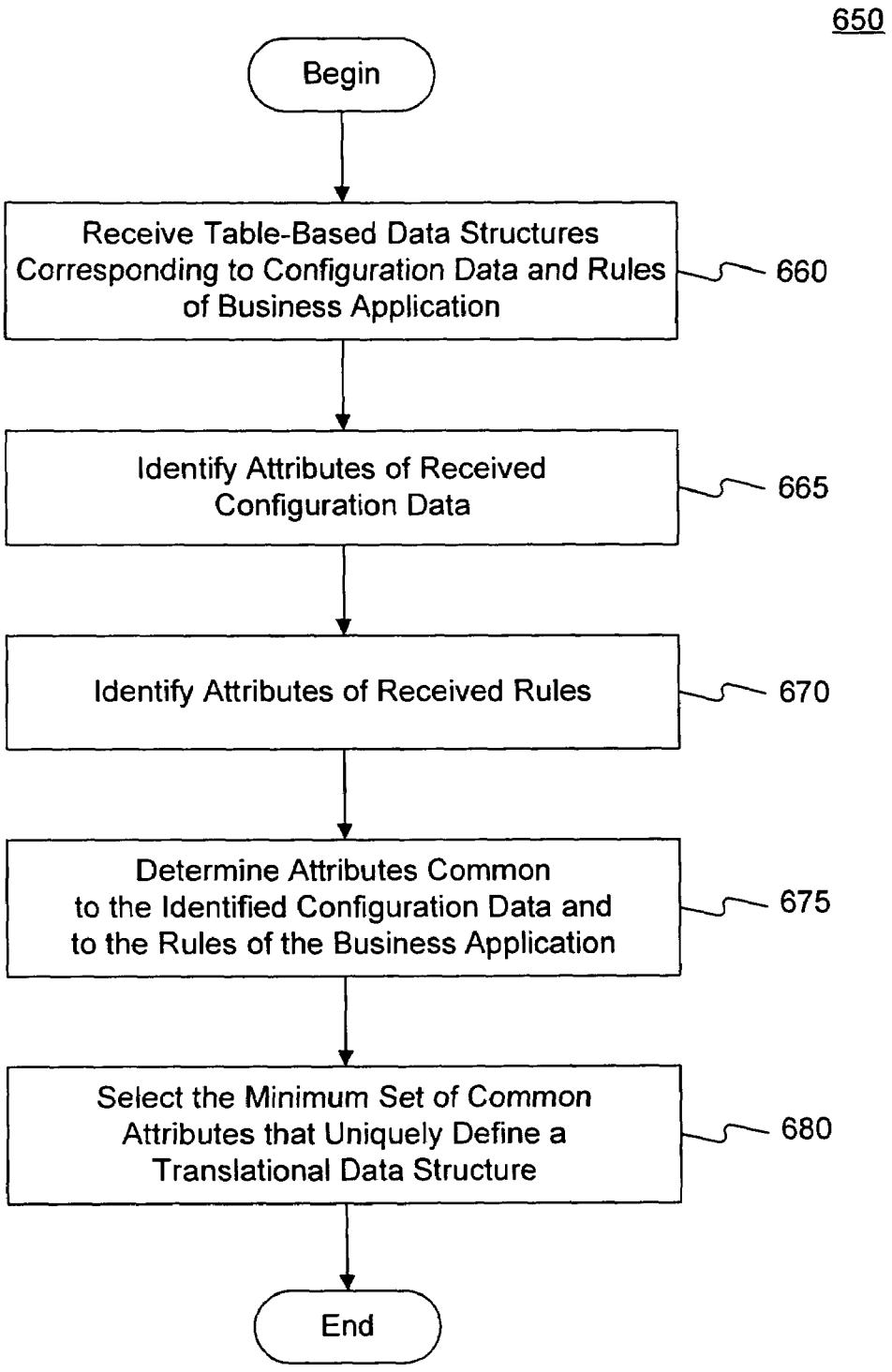
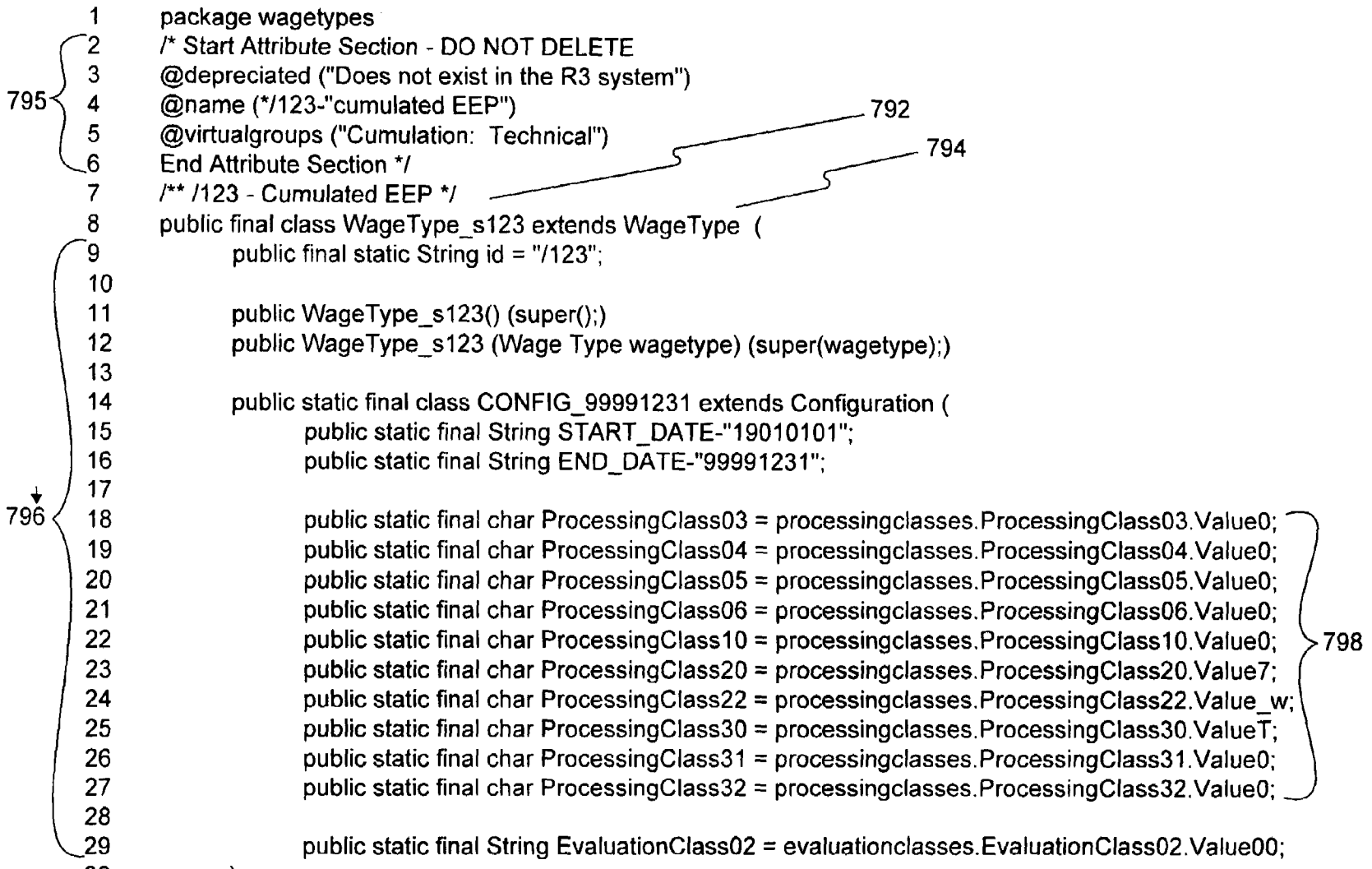

650
Begin
Receive Table-Based Data Structures Corresponding to Configuration Data and Rules of Business Application
660
Identify Attributes of Received Configuration Data
665
Identify Attributes of Received Rules
670
Determine Attributes Common to the Identified Configuration Data and to the Rules of the Business Application
675
Select the Minimum Set of Common Attributes that Uniquely Define a Translational Data Structure
680
End

FIGURE 6B




```
1   package wagetypes
2   /* Start Attribute Section - DO NOT DELETE
3   @depreciated ("Does not exist in the R3 system")
4   @name (*/123-"cumulated EEP")
5   @virtualgroups ("Cumulation:  Technical")
6   End Attribute Section */
7   /** /123 - Cumulated EEP */
8   public final class WageType_s123 extends WageType  (
9           public final static String id = "/123";
10
11          public WageType_s123() (super();)
12          public WageType_s123 (Wage Type wagetype) (super(wagetype);)
13
14          public static final class CONFIG_99991231 extends Configuration (
15                  public static final String START_DATE-"19010101";
16                  public static final String END_DATE-"99991231";
17
18                  public static final char ProcessingClass03 = processingclasses.ProcessingClass03.Value0;
19                  public static final char ProcessingClass04 = processingclasses.ProcessingClass04.Value0;
20                  public static final char ProcessingClass05 = processingclasses.ProcessingClass05.Value0;
21                  public static final char ProcessingClass06 = processingclasses.ProcessingClass06.Value0;
22                  public static final char ProcessingClass10 = processingclasses.ProcessingClass10.Value0;
23                  public static final char ProcessingClass20 = processingclasses.ProcessingClass20.Value7;
24                  public static final char ProcessingClass22 = processingclasses.ProcessingClass22.Value_w;
25                  public static final char ProcessingClass30 = processingclasses.ProcessingClass30.ValueT;
26                  public static final char ProcessingClass31 = processingclasses.ProcessingClass31.Value0;
27                  public static final char ProcessingClass32 = processingclasses.ProcessingClass32.Value0;
28
29                  public static final String EvaluationClass02 = evaluationclasses.EvaluationClass02.Value00;
30          )
31  )
32
```

FIGURE 7A

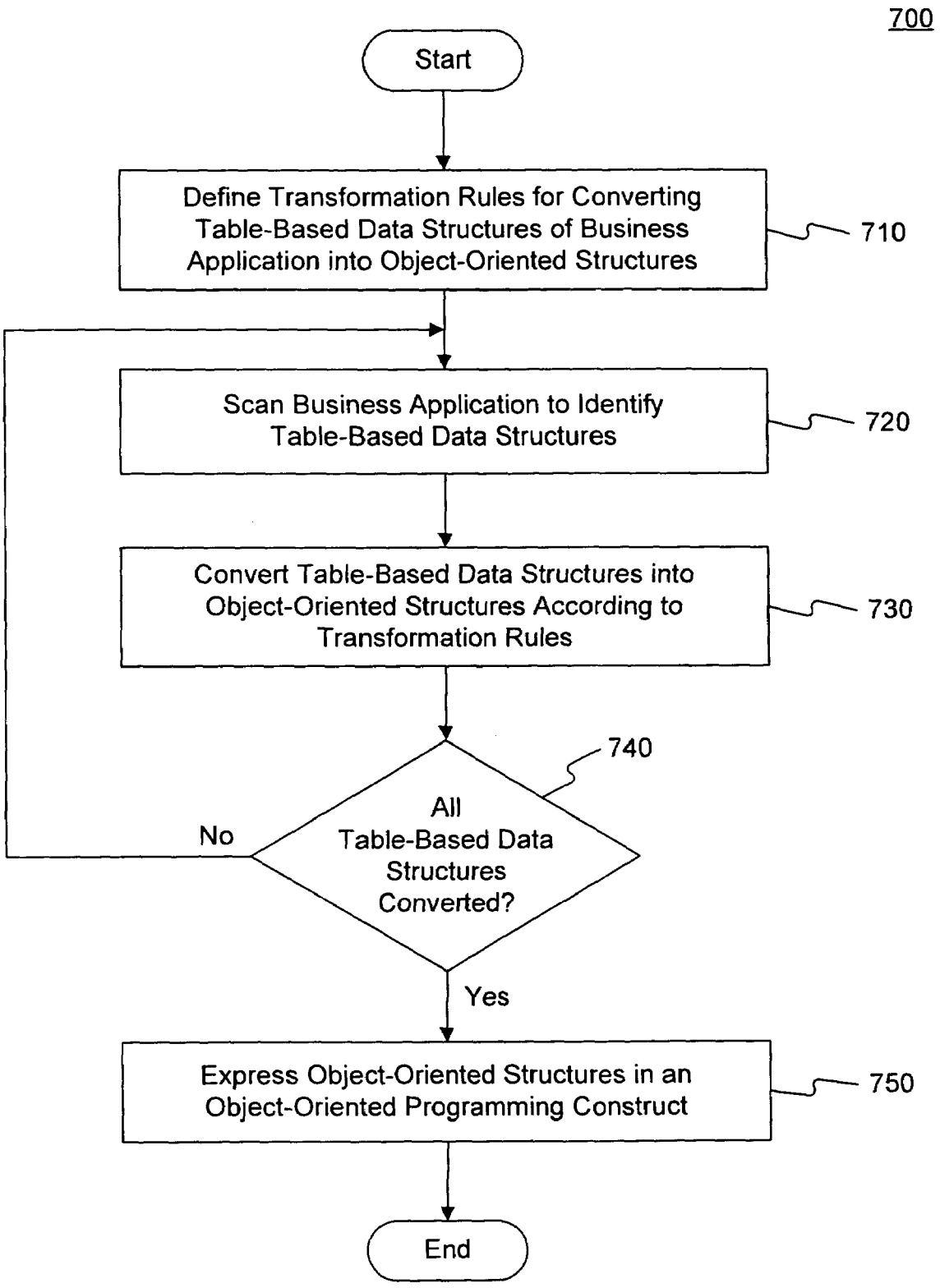

700
Start
Define Transformation Rules for Converting Table-Based Data Structures of Business Application into Object-Oriented Structures
710
Scan Business Application to Identify Table-Based Data Structures
720
Convert Table-Based Data Structures into Object-Oriented Structures According to Transformation Rules
730
740
All Table-Based Data Structures Converted?
No
Yes
Express Object-Oriented Structures in an Object-Oriented Programming Construct
750
End

FIGURE 7B

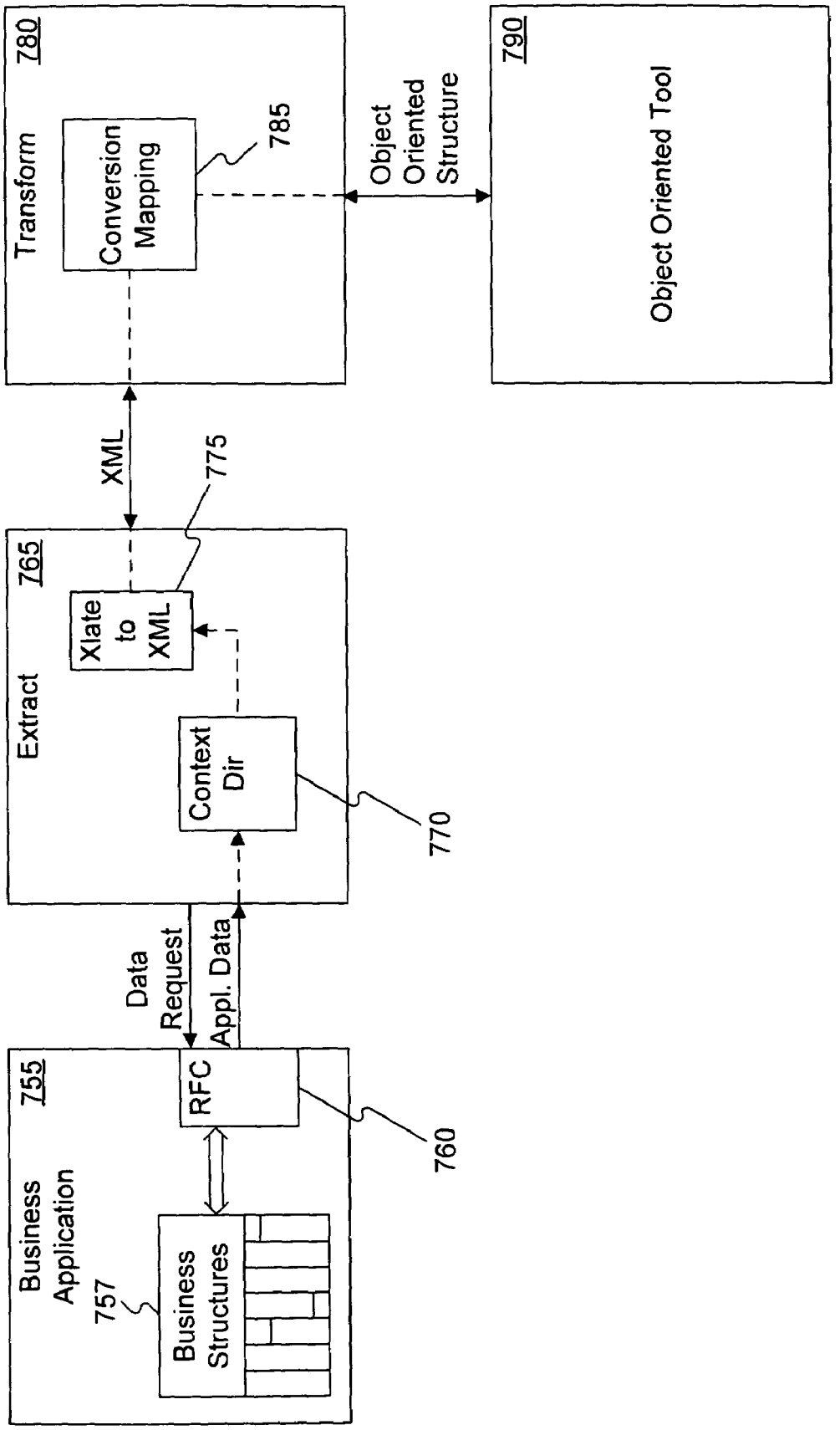

780
Transform
Conversion Mapping
785
Object Oriented Structure
790
Object Oriented Tool
XML
775
765
Extract
Xlate to XML
Context Dir
770
Data Request
Appl. Data
755
Business Application
757
Business Structures
RFC
760

FIGURE 7C

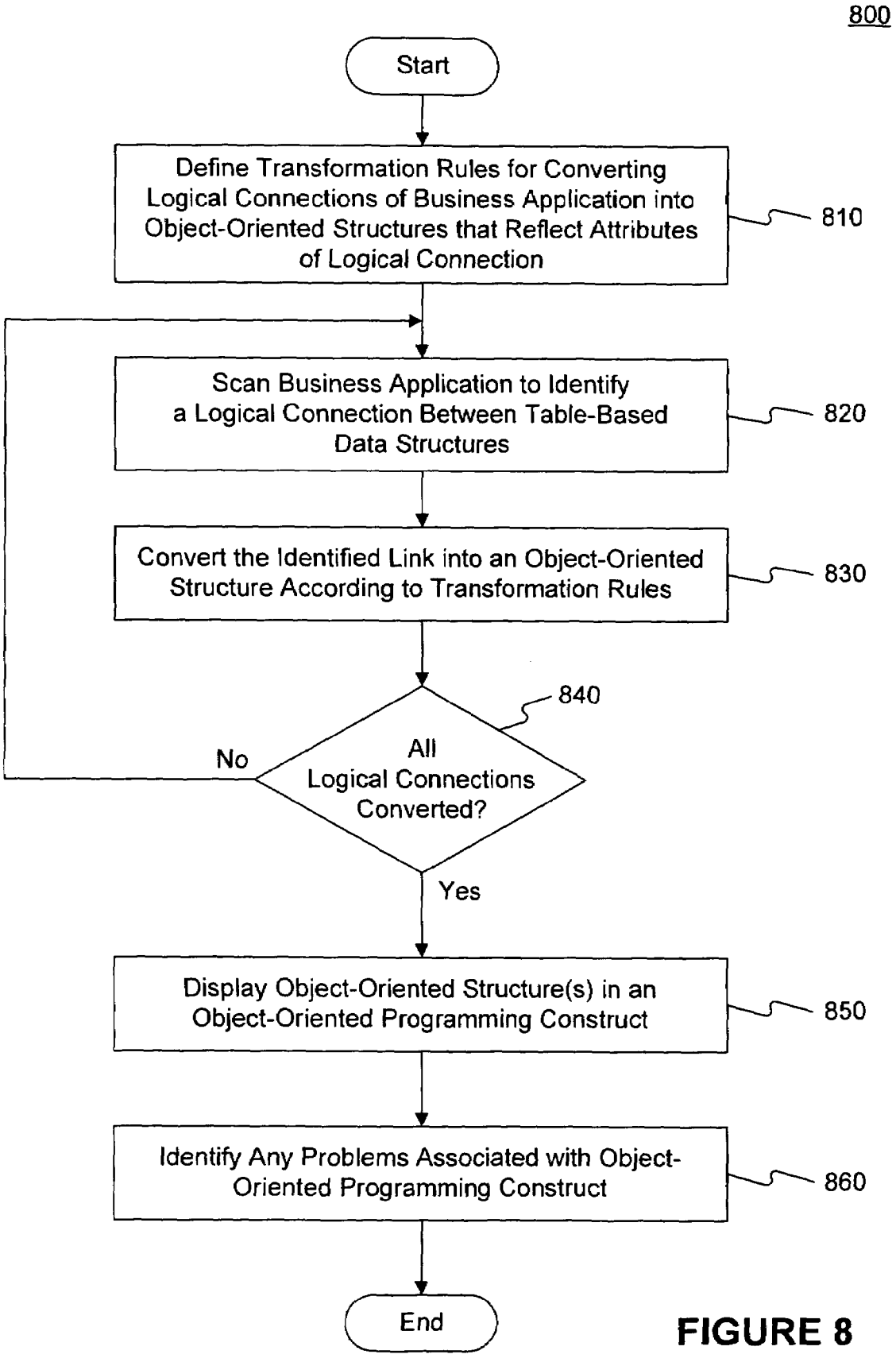

800
Start
Define Transformation Rules for Converting Logical Connections of Business Application into Object-Oriented Structures that Reflect Attributes of Logical Connection
810
Scan Business Application to Identify a Logical Connection Between Table-Based Data Structures
820
Convert the Identified Link into an Object-Oriented Structure According to Transformation Rules
830
840
All Logical Connections Converted?
No
Yes
Display Object-Oriented Structure(s) in an Object-Oriented Programming Construct
850
Identify Any Problems Associated with Object-Oriented Programming Construct
860
End

FIGURE 8

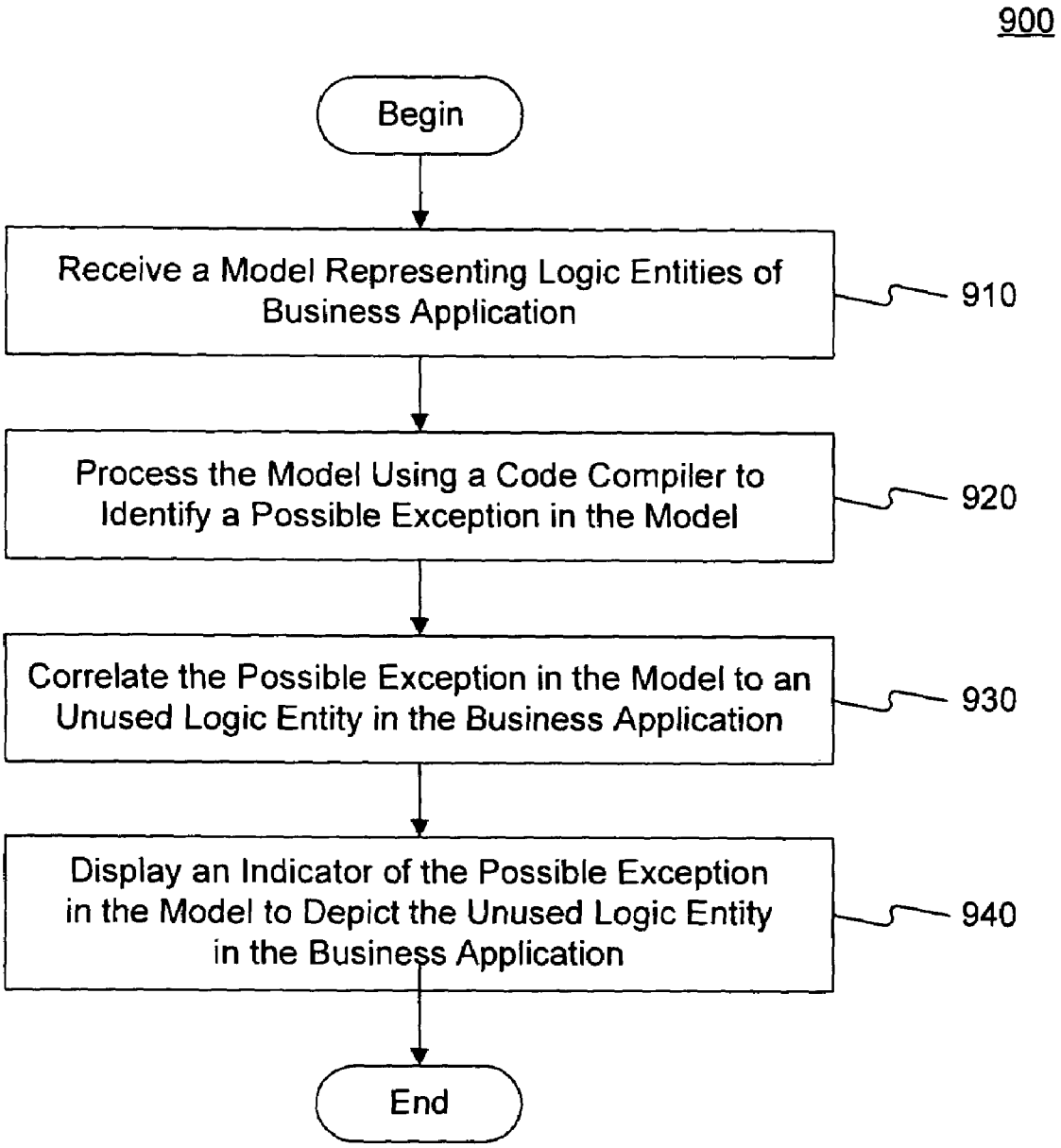

900
Begin
Receive a Model Representing Logic Entities of Business Application
910
Process the Model Using a Code Compiler to Identify a Possible Exception in the Model
920
Correlate the Possible Exception in the Model to an Unused Logic Entity in the Business Application
930
Display an Indicator of the Possible Exception in the Model to Depict the Unused Logic Entity in the Business Application
940
End

FIGURE 9

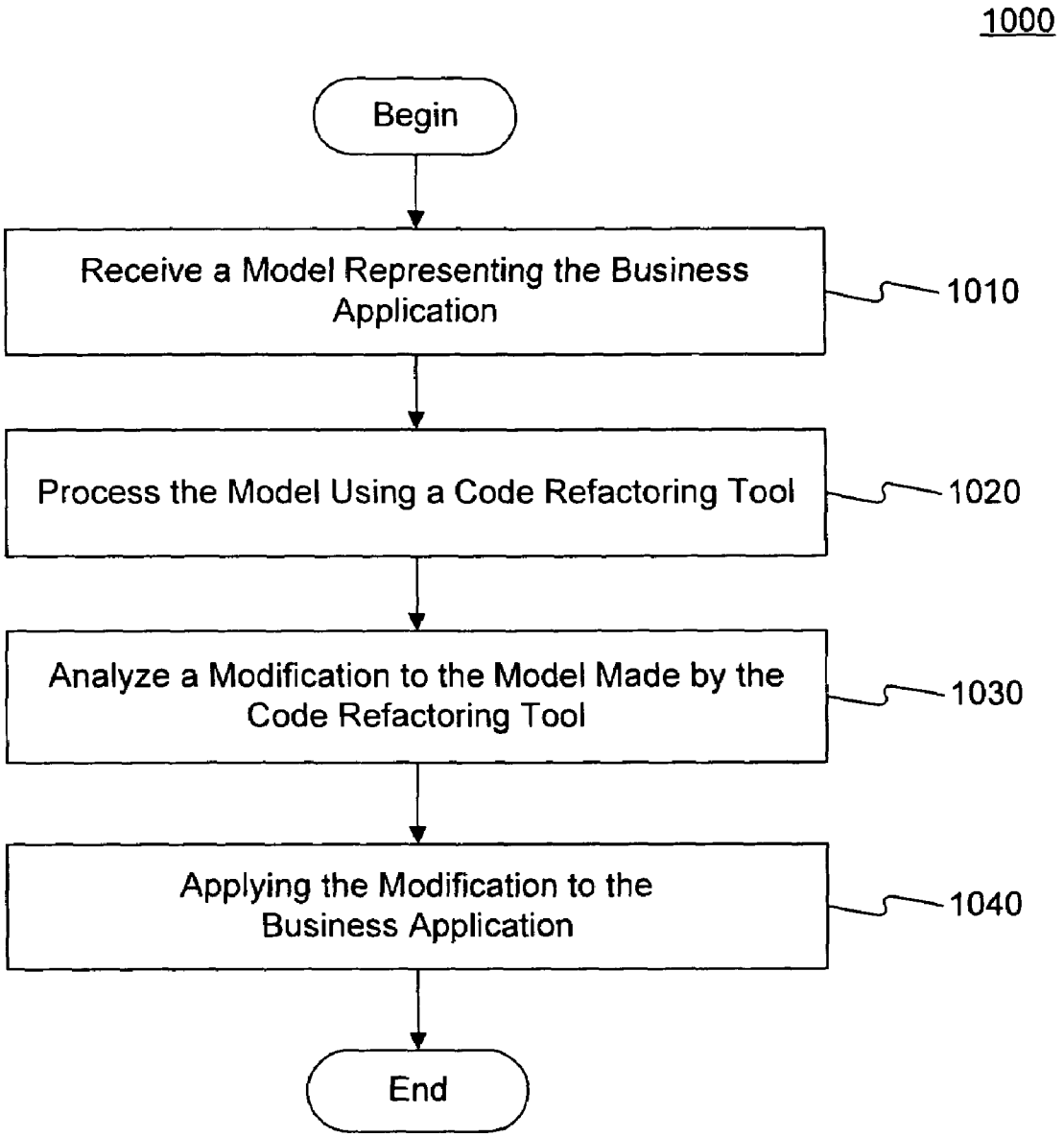

1000
Begin
Receive a Model Representing the Business Application
1010
Process the Model Using a Code Refactoring Tool
1020
Analyze a Modification to the Model Made by the Code Refactoring Tool
1030
Applying the Modification to the Business Application
1040
End

FIGURE 10

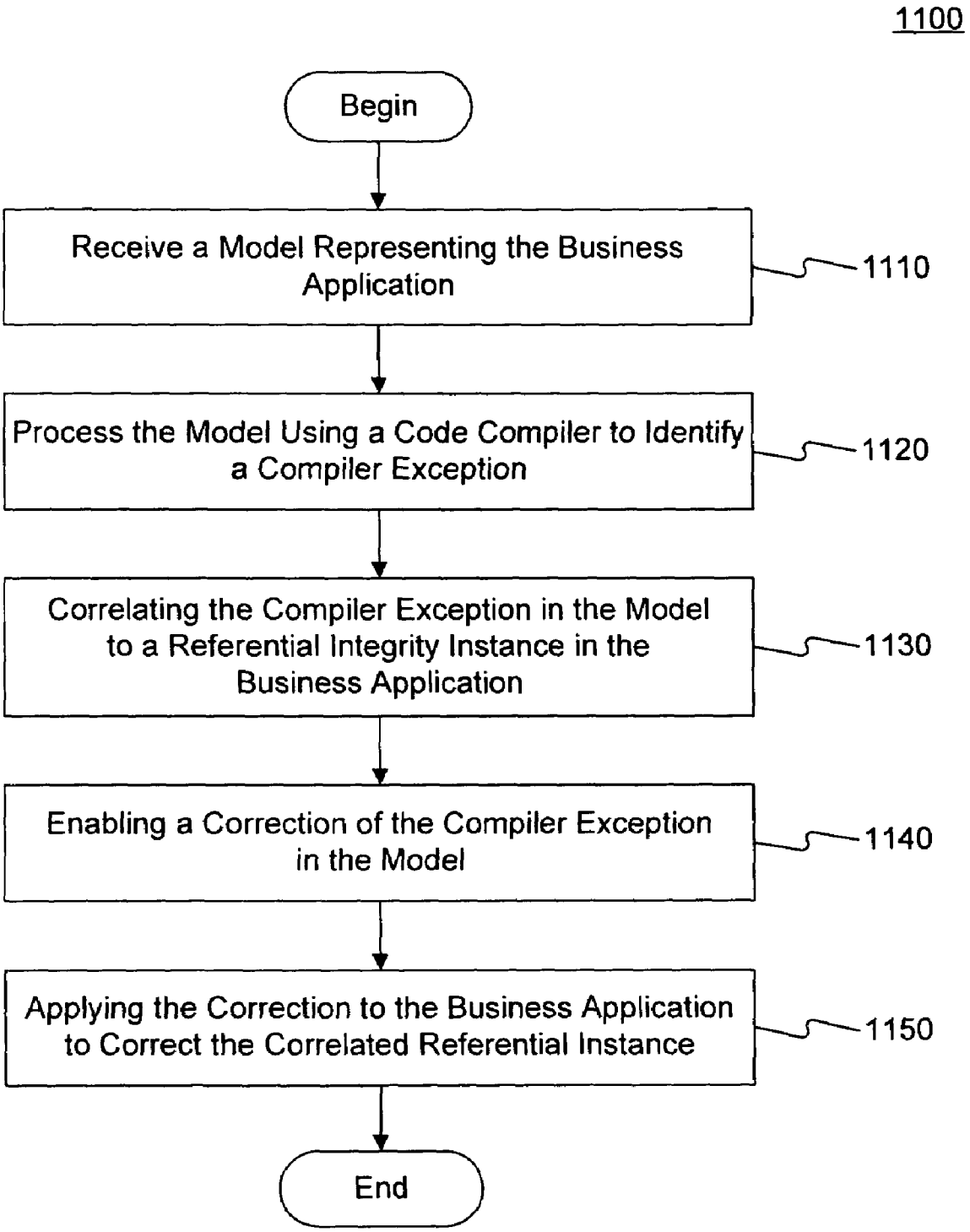

1100
Begin
Receive a Model Representing the Business Application
1110
Process the Model Using a Code Compiler to Identify a Compiler Exception
1120
Correlating the Compiler Exception in the Model to a Referential Integrity Instance in the Business Application
1130
Enabling a Correction of the Compiler Exception in the Model
1140
Applying the Correction to the Business Application to Correct the Correlated Referential Instance
1150
End

FIGURE 11

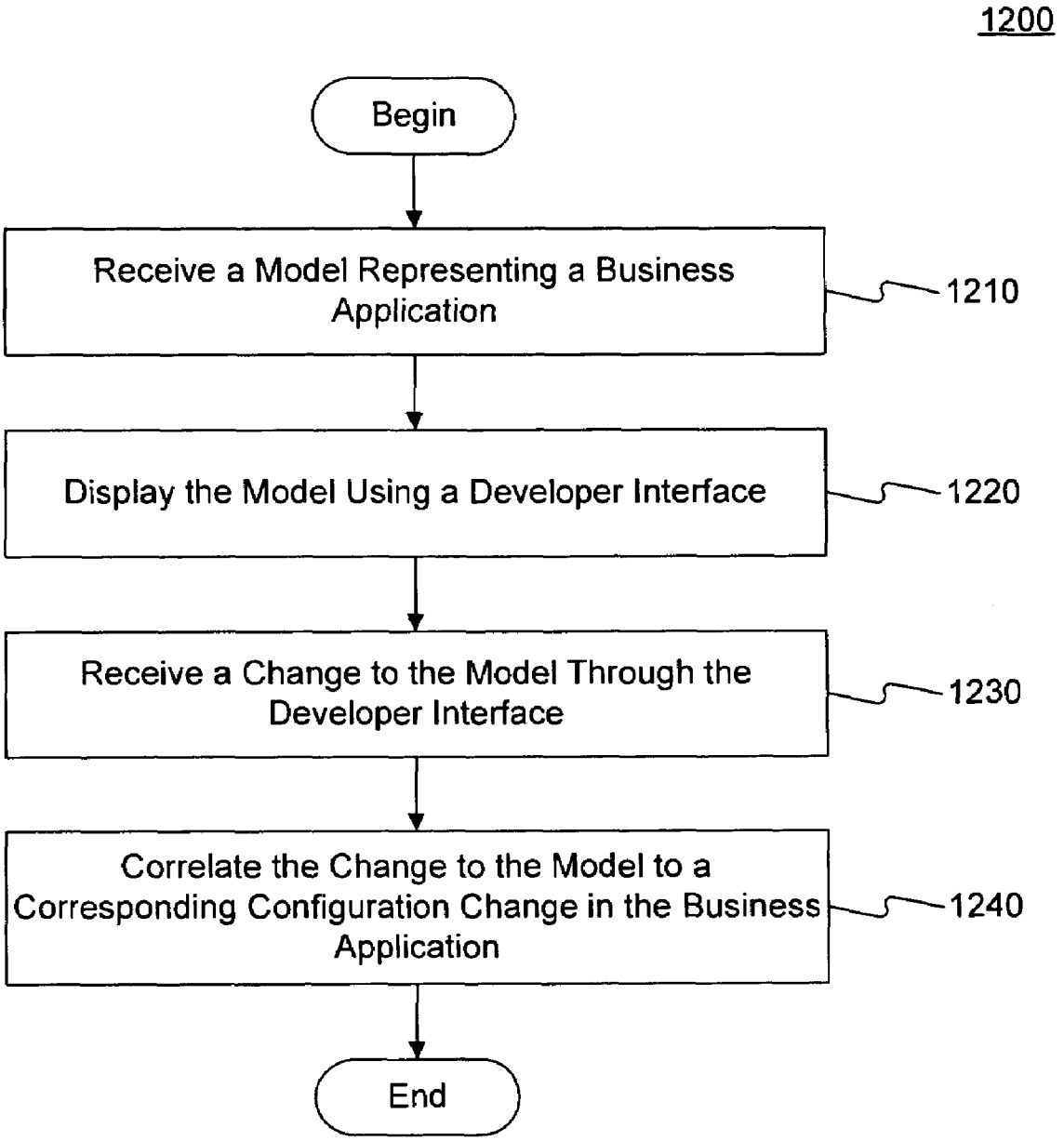

1200
Begin
Receive a Model Representing a Business Application
1210
Display the Model Using a Developer Interface
1220
Receive a Change to the Model Through the Developer Interface
1230
Correlate the Change to the Model to a Corresponding Configuration Change in the Business Application
1240
End

FIGURE 12

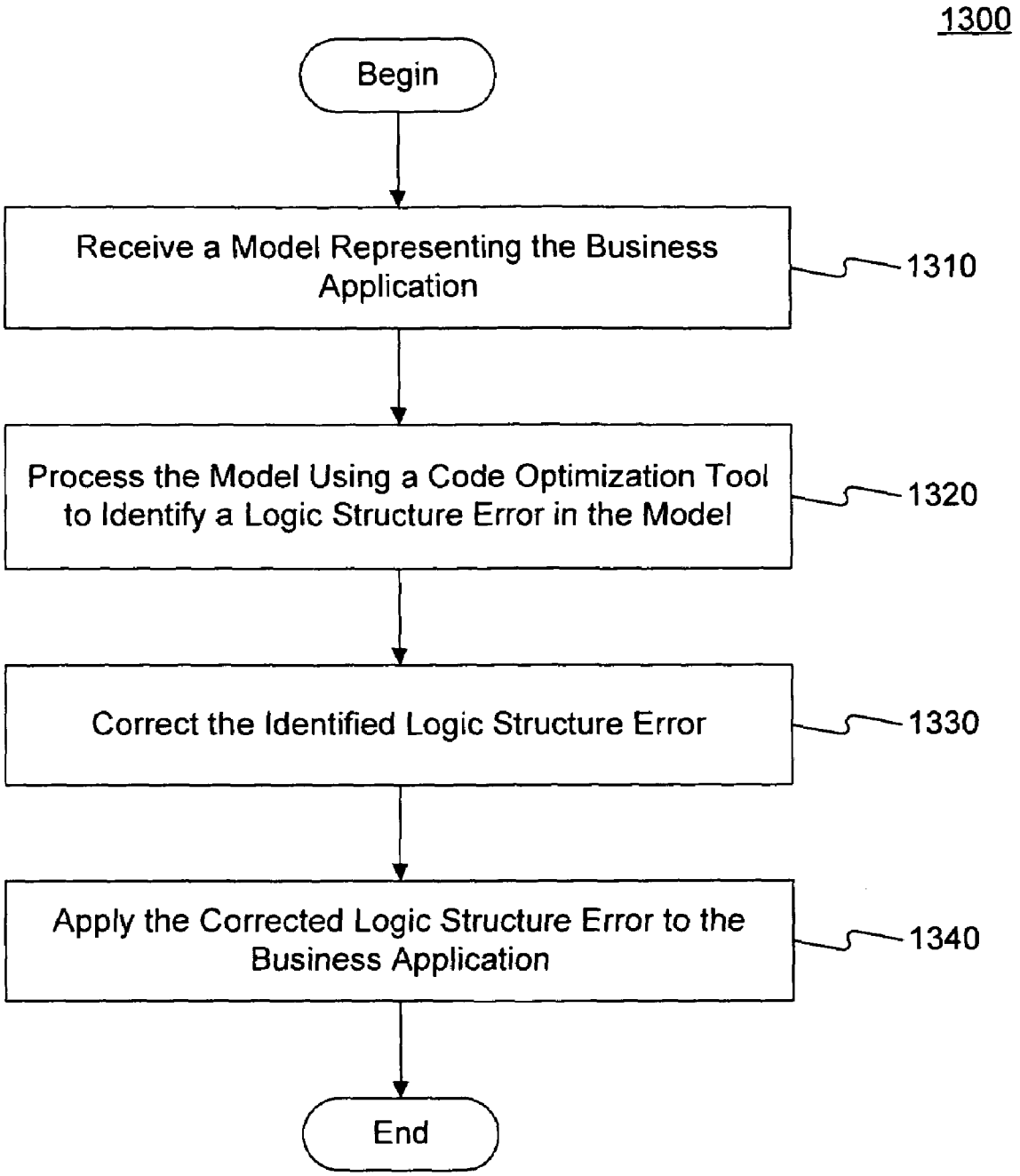

1300
Begin
Receive a Model Representing the Business Application
1310
Process the Model Using a Code Optimization Tool to Identify a Logic Structure Error in the Model
1320
Correct the Identified Logic Structure Error
1330
Apply the Corrected Logic Structure Error to the Business Application
1340
End

FIGURE 13

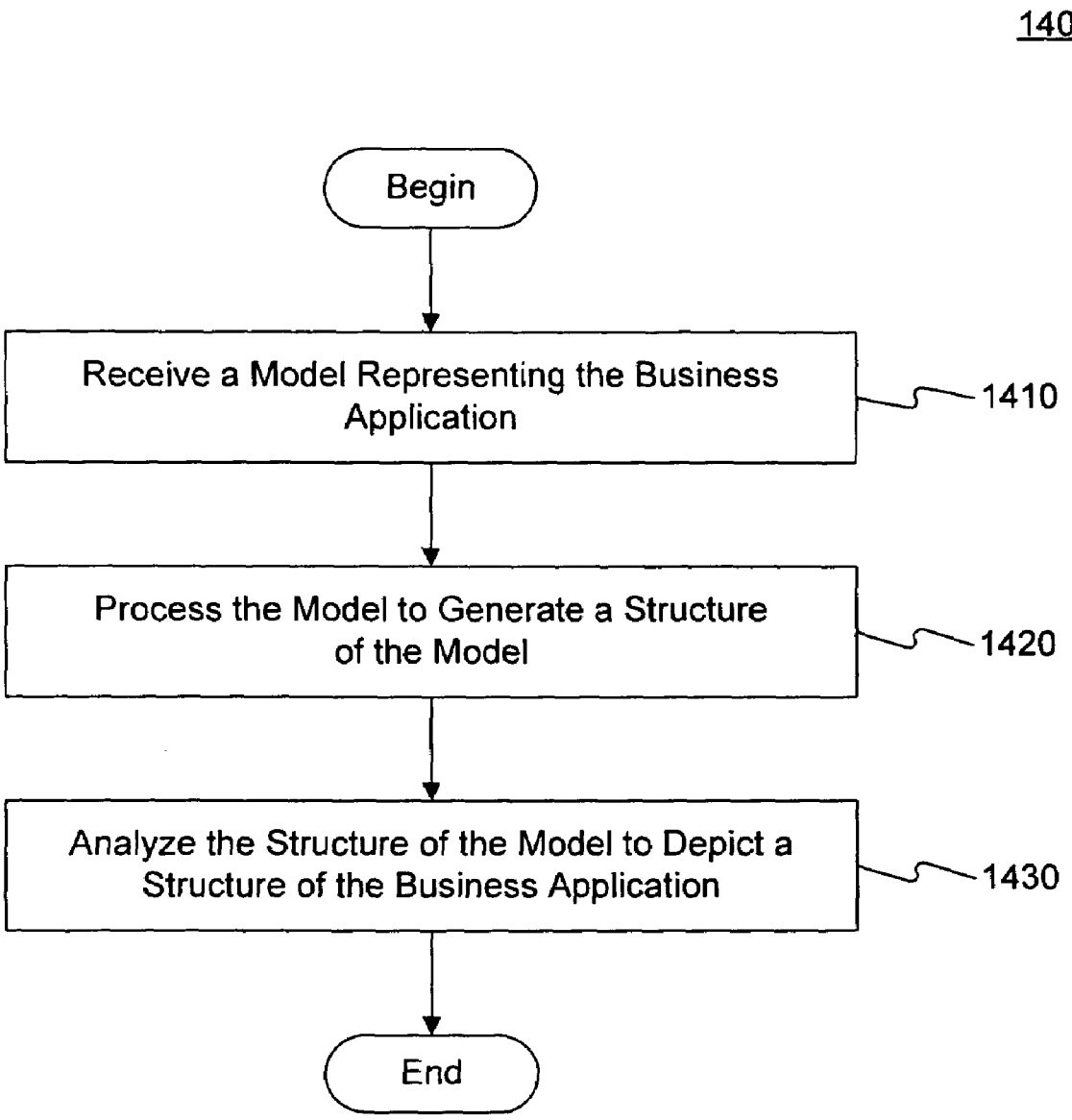

1400
Begin
Receive a Model Representing the Business Application
1410
Process the Model to Generate a Structure of the Model
1420
Analyze the Structure of the Model to Depict a Structure of the Business Application
1430
End

FIGURE 14

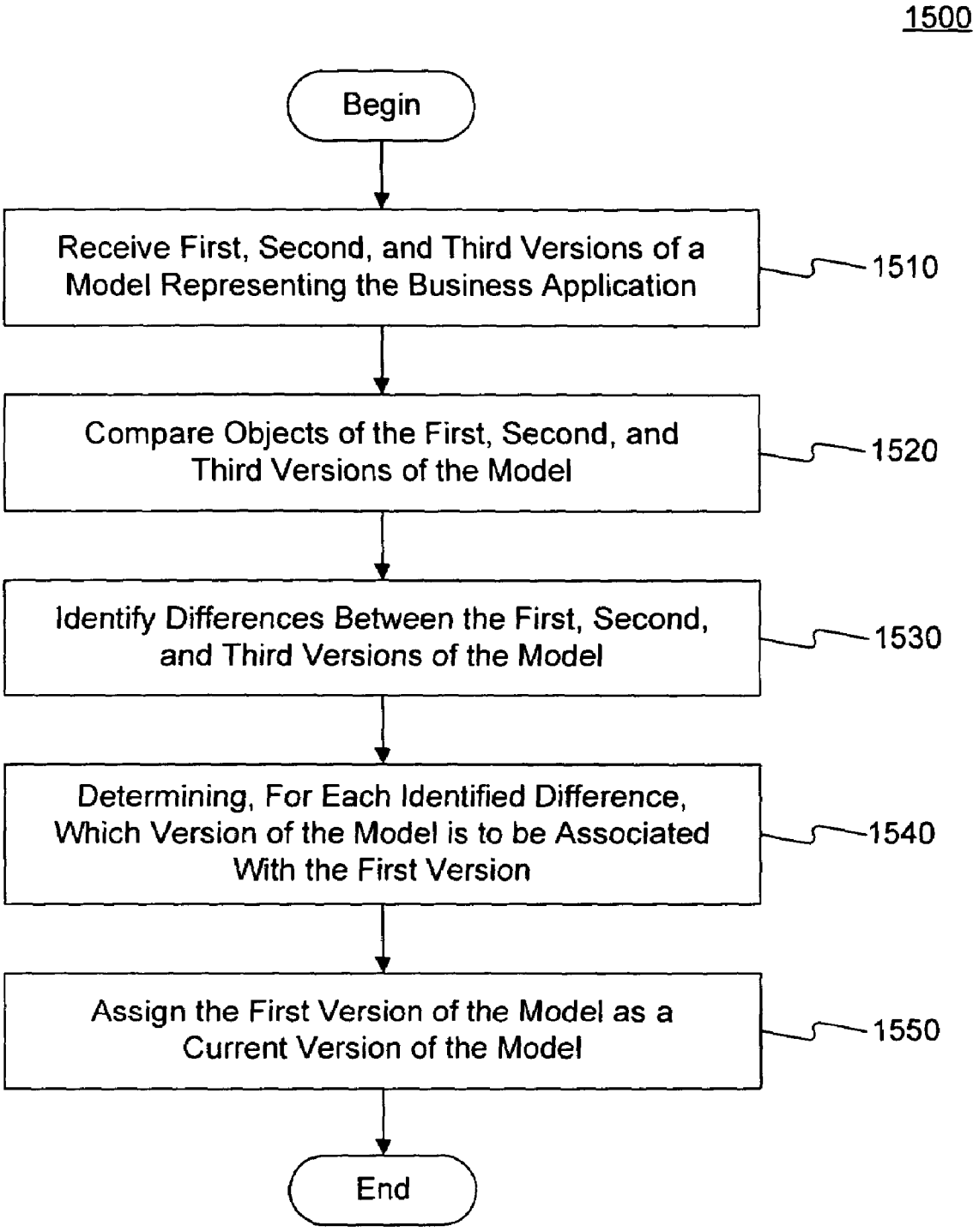

1500
Begin
Receive First, Second, and Third Versions of a Model Representing the Business Application
1510
Compare Objects of the First, Second, and Third Versions of the Model
1520
Identify Differences Between the First, Second, and Third Versions of the Model
1530
Determining, For Each Identified Difference, Which Version of the Model is to be Associated With the First Version
1540
Assign the First Version of the Model as a Current Version of the Model
1550
End

FIGURE 15

1600

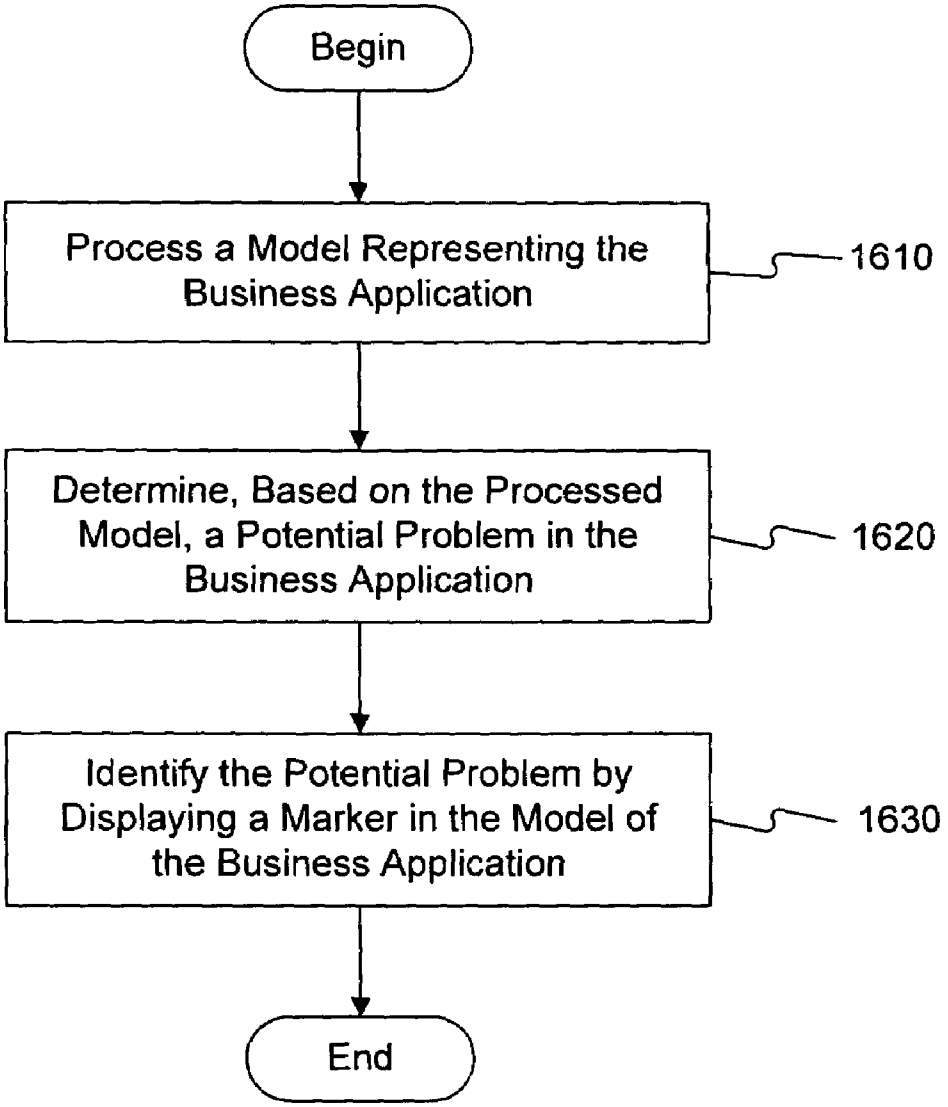


FIGURE 16

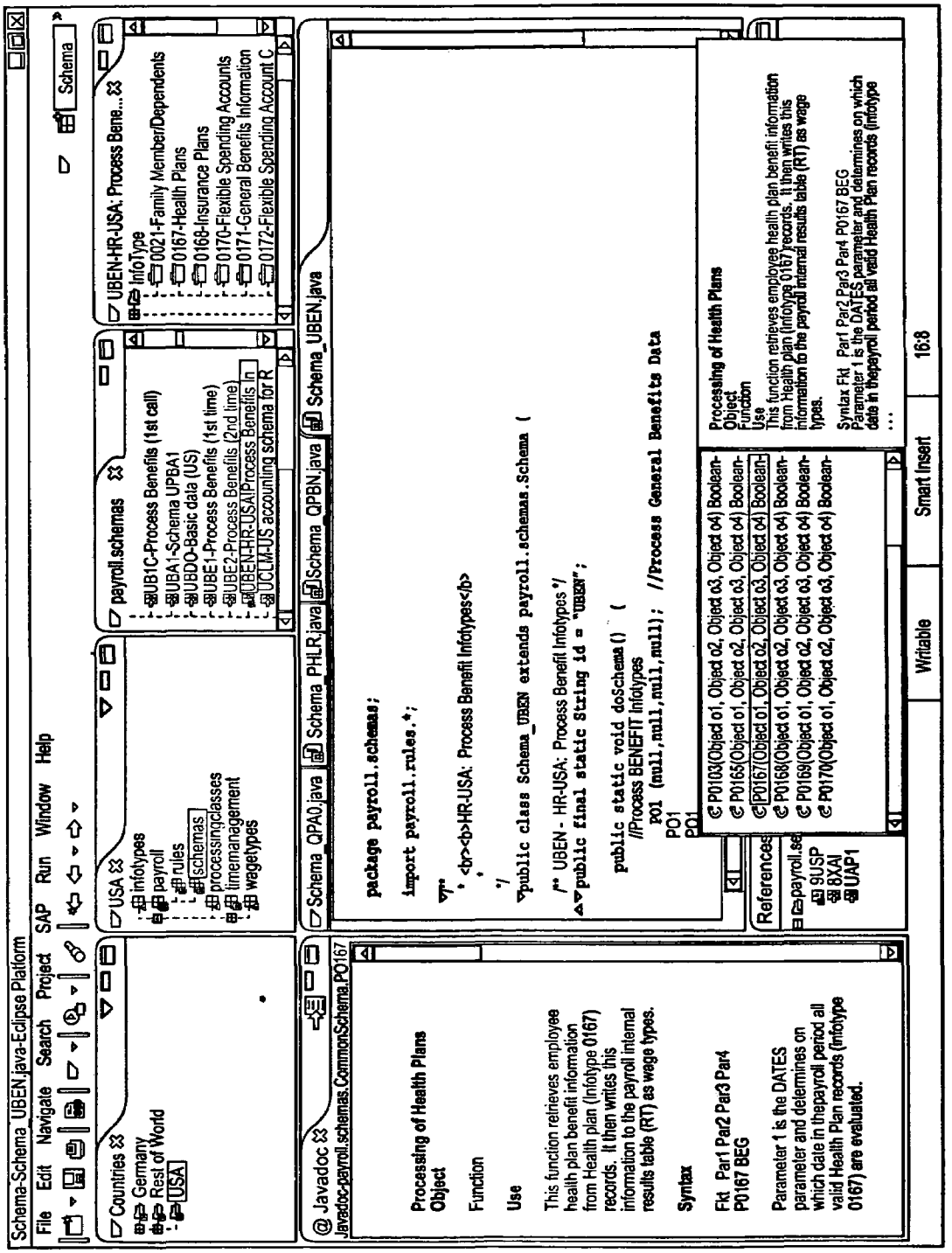

Schema-Schema_UBEN.java-Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
Schema
Countries
Germany
Rest of World
USA
USA
infotypes
payroll
rules
schemas
processingclasses
timemanagement
wagetypes
payroll.schemas
UB1C-Process Benefits (1st call)
UBA1-Schema UPBA1
UBDO-Basic data (US)
UBE1-Process Benefits (1st time)
UBE2-Process Benefits (2nd time)
UBEN-HR-USA:Process Benefits In
UCLM-US accounting schema for R
UBEN-HR-USA: Process Bene...
InfoType
0021-Family Member/Dependents
0167-Health Plans
0168-Insurance Plans
0170-Flexible Spending Accounts
0171-General Benefits Information
0172-Flexible Spending Account C
Javadoc
Javadoc-payroll.schemas.CommonSchema.P0167
Processing of Health Plans
Object
Function
Use
This function retrieves employee health plan benefit information from Health plan (infotype 0167) records. It then writes this information to the payroll internal results table (RT) as wage types.
Syntax
Fkt Par1 Par2 Par3 Par4
P0167 BEG
Parameter 1 is the DATES parameter and determines on which date in thepayroll period all valid Health Plan records (infotype 0167) are evaluated.
Schema_QPA0.java
Schema_PHLR.java
Schema_QPBN.java
Schema_UBEN.java
package payroll.schemas;
import payroll.rules.*;
/**
*
HR-USA: Process Benefit Infotypes
*
*/
public class Schema_UBEN extends payroll.schemas.Schema {
/** UBEN - HR-USA: Process Benefit Infotypes */
public final static String id = "UBEN";
public static void doSchema() {
//Process BENEFIT Infotypes
P01 (null,null,null,null); //Process General Benefits Data
P01
P01
P0103(Object o1, Object o2, Object o3, Object o4) Boolean-
P0165(Object o1, Object o2, Object o3, Object o4) Boolean-
P0167(Object o1, Object o2, Object o3, Object o4) Boolean-
P0168(Object o1, Object o2, Object o3, Object o4) Boolean-
P0169(Object o1, Object o2, Object o3, Object o4) Boolean-
P0170(Object o1, Object o2, Object o3, Object o4) Boolean-
References
payroll.sc
9USP
8XAI
UAP1
Processing of Health Plans
Object
Function
Use
This function retrieves employee health plan benefit information from Health plan (infotype 0167) records. It then writes this information to the payroll internal results table (RT) as wage types.
Syntax Fkt Par1 Par2 Par3 Par4 P0167 BEG
Parameter 1 is the DATES parameter and determines on which date in thepayroll period all valid Health Plan records (infotype
...
Writable
Smart Insert
16:8

FIG. 17

Countries
Main project: USA.rvc.800
Argentina
Australia
Austria
Belgium
Brazil
Canada
China
Denmark
Finland
France
Germany
Great Britain
Hong Kong
India
Indonesia
Ireland
Italy
Japan
Malaysia
Mexico
Netherlands
New Zealand
Norway
Phillipines
Portugal
Rest of world
Singapore
South Africa
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
USA
ma
800
rvc
>800
Venezuela

FIG. 18

Countries
Main project: USA.rvc.800
ma
800
Argentina
Australia
Austria
Belgium
Brazil
Canada
China
Denmark
Finland
France
Germany
Great Britain
Hong Kong
India
Indonesia
Ireland
Italy
Japan
Malaysia
Mexico
Netherlands
New Zealand
Norway
Phillipines
Portugal
Rest of world
Singapore
South Africa
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
USA
Venezuela
rvc
>800
Argentina
Australia
Austria
Belgium
Brazil

FIG. 19

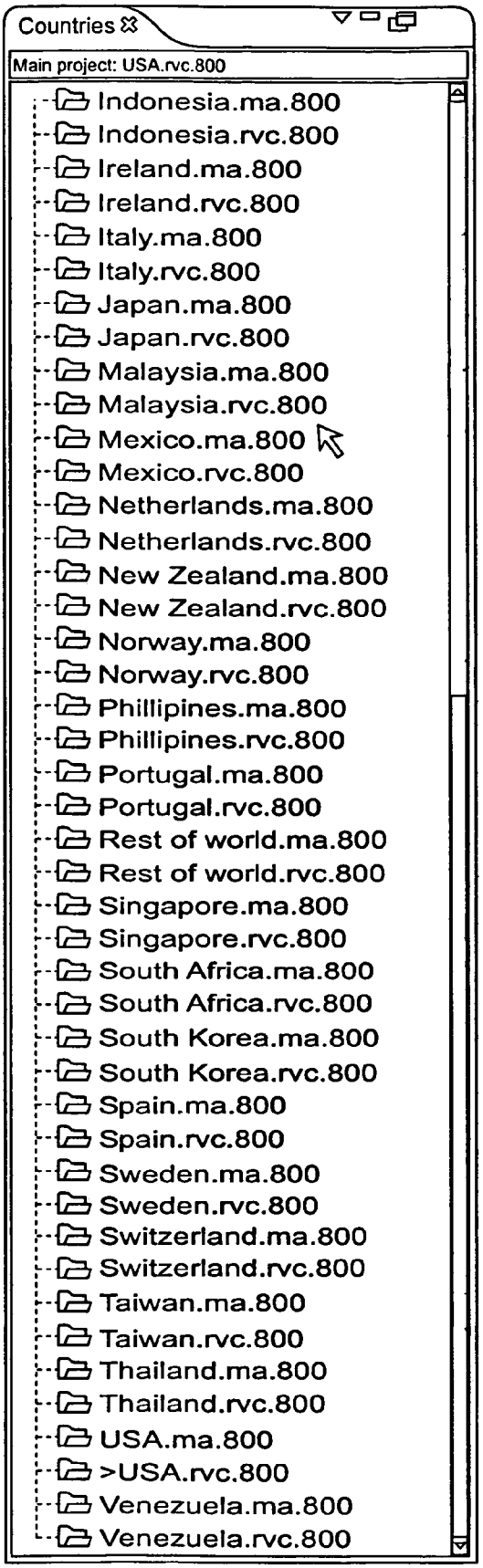

Countries
Main project: USA.rvc.800
Indonesia.ma.800
Indonesia.rvc.800
Ireland.ma.800
Ireland.rvc.800
Italy.ma.800
Italy.rvc.800
Japan.ma.800
Japan.rvc.800
Malaysia.ma.800
Malaysia.rvc.800
Mexico.ma.800
Mexico.rvc.800
Netherlands.ma.800
Netherlands.rvc.800
New Zealand.ma.800
New Zealand.rvc.800
Norway.ma.800
Norway.rvc.800
Phillipines.ma.800
Phillipines.rvc.800
Portugal.ma.800
Portugal.rvc.800
Rest of world.ma.800
Rest of world.rvc.800
Singapore.ma.800
Singapore.rvc.800
South Africa.ma.800
South Africa.rvc.800
South Korea.ma.800
South Korea.rvc.800
Spain.ma.800
Spain.rvc.800
Sweden.ma.800
Sweden.rvc.800
Switzerland.ma.800
Switzerland.rvc.800
Taiwan.ma.800
Taiwan.rvc.800
Thailand.ma.800
Thailand.rvc.800
USA.ma.800
>USA.rvc.800
Venezuela.ma.800
Venezuela.rvc.800

FIG. 20

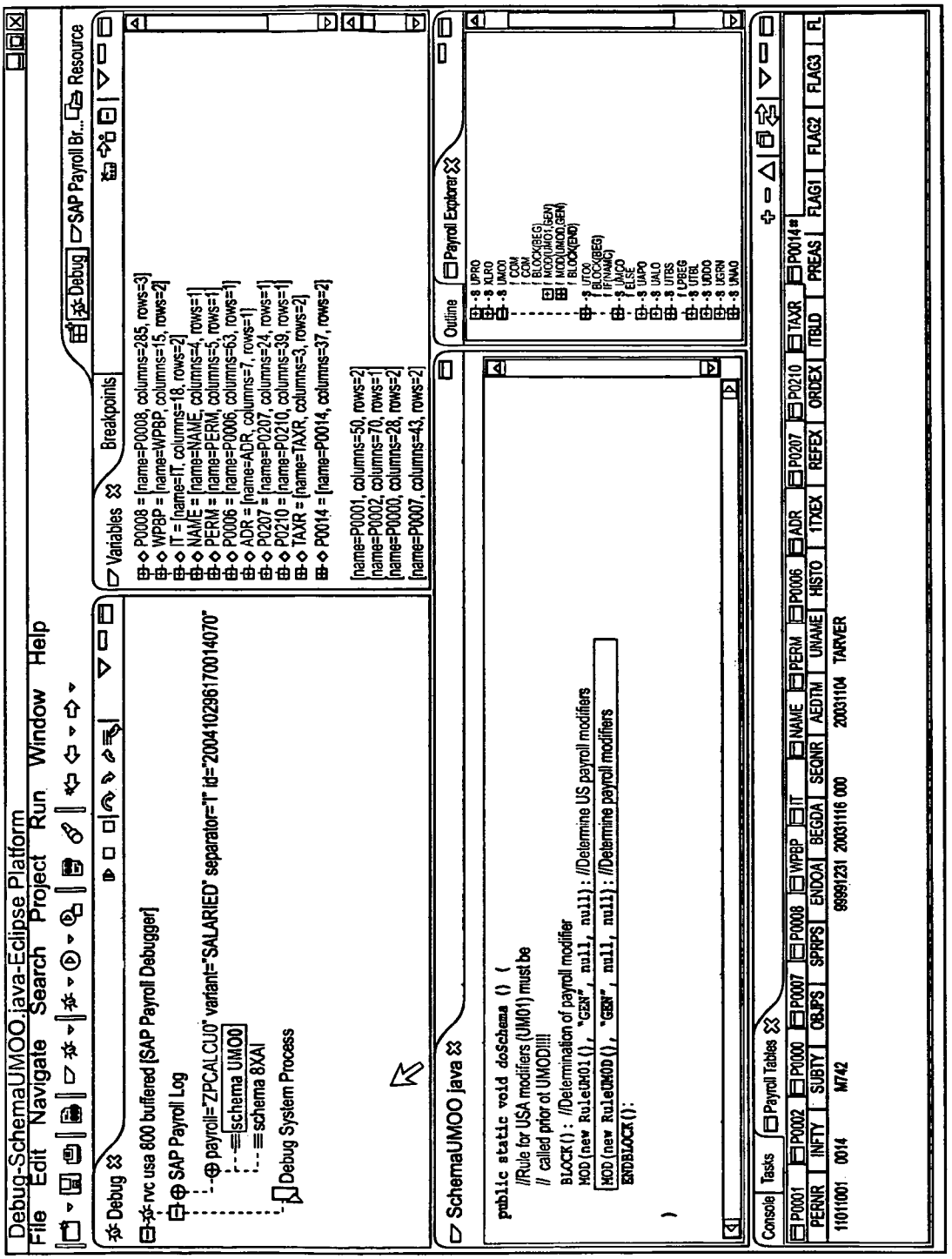

Debug-SchemaUMOO.java-Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
rvc usa 800 buffered [SAP Payroll Debugger]
SAP Payroll Log
payroll="ZPCALCU0" variant="SALARIED" separator="|" id="20041029617014070"
schema UM00
schema 8XAI
Debug System Process
Variables
Breakpoints
SAP Payroll Br...
Resource
P0008 = [name=P0008, columns=285, rows=3]
WPBP = [name=WPBP, columns=15, rows=2]
IT = [name=IT, columns=18, rows=2]
NAME = [name=NAME, columns=4, rows=1]
PERM = [name=PERM, columns=5, rows=1]
P0006 = [name=P0006, columns=63, rows=1]
ADR = [name=ADR, columns=7, rows=1]
P0207 = [name=P0207, columns=24, rows=1]
P0210 = [name=P0210, columns=39, rows=1]
TAXR = [name=TAXR, columns=3, rows=2]
P0014 = [name=P0014, columns=37, rows=2]
[name=P0001, columns=50, rows=2]
[name=P0002, columns=70, rows=1]
[name=P0000, columns=28, rows=2]
[name=P0007, columns=43, rows=2]
SchemaUMOO.java
public static void doSchema () {
//Rule for USA modifiers (UM01) must be
// called prior ot UMOD!!!!
BLOCK(1) : //Determination of payroll modifier
MOD(new RuleUM01(), "GEN", null, null) : //Determine US payroll modifiers
MOD(new RuleUMOD(), "GEN", null, null) : //Determine payroll modifiers
ENDBLOCK(1):
}
Outline
Payroll Explorer
UPRO
KLRO
UMOO
COM
COM
BLOCK(BEG)
MOD(UM01,GEN)
MOD(UMOD,GEN)
BLOCK(END)
UTOI
BLOCK(BEG)
PPWMC)
UGPO
UEEX
UAPO
UALO
UTBS
LPBEG
UTBL
UDDO
UGRN
UNAO
Console
Tasks
Payroll Tables
P0001
P0002
P0000
P0007
P0008
WPBP
IT
NAME
PERM
P0006
ADR
P0207
P0210
TAXR
P0014
PERNR
INFTY
SUBTY
OBJPS
SPRPS
ENDDA
BEGDA
SEQNR
AEDTM
UNAME
HISTO
ITXEX
REFEX
ORDEX
ITBLD
PREAS
FLAG1
FLAG2
FLAG3
FL
11011001 0014 M742 99991231 20031116 000 20031104 TARVER

FIG. 21

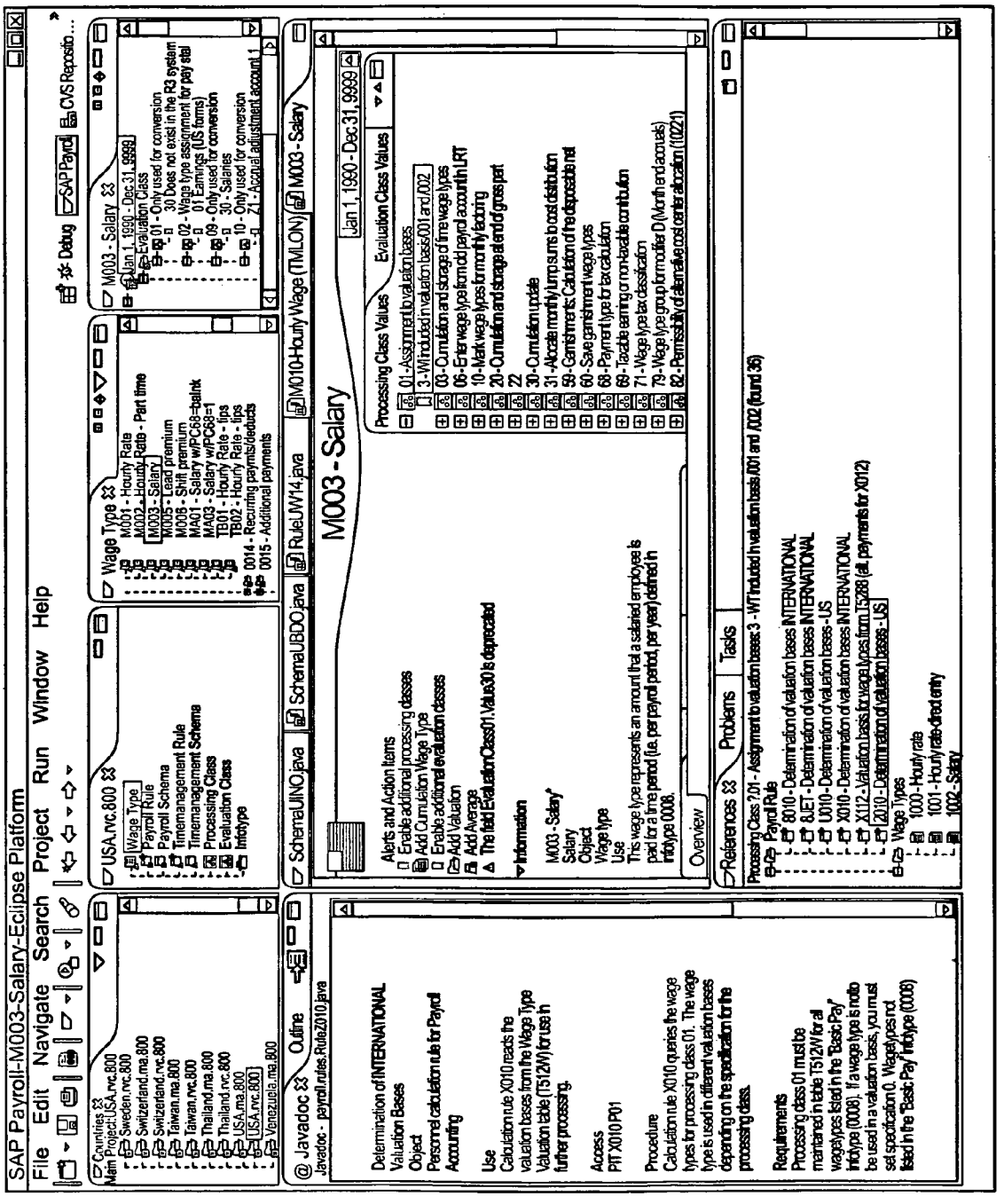

SAP Payroll-M003-Salary-Eclipse Platform
File Edit Navigate Search Project Run Window Help
M003 - Salary
M003 - Salary
Determination of INTERNATIONAL
Valuation Bases
Object
Personnel calculation rule for Payroll
Accounting
Use
Calculation rule X010 reads the valuation bases from the Wage Type Valuation table (T512W) for use in further processing.
Access
PIT X010 P01
Procedure
Calculation rule X010 queries the wage types for processing class 01. The wage type is used in different valuation bases depending on the specification for the processing class.
Requirements
Processing class 01 must be maintained in table T512W for all wagetypes listed in the 'Basic Pay' infotype (0008). If a wage type is not to be used in a valuation basis, you must set specification 0. Wagetypes not listed in the 'Basic Pay' infotype (0008)

FIG. 22

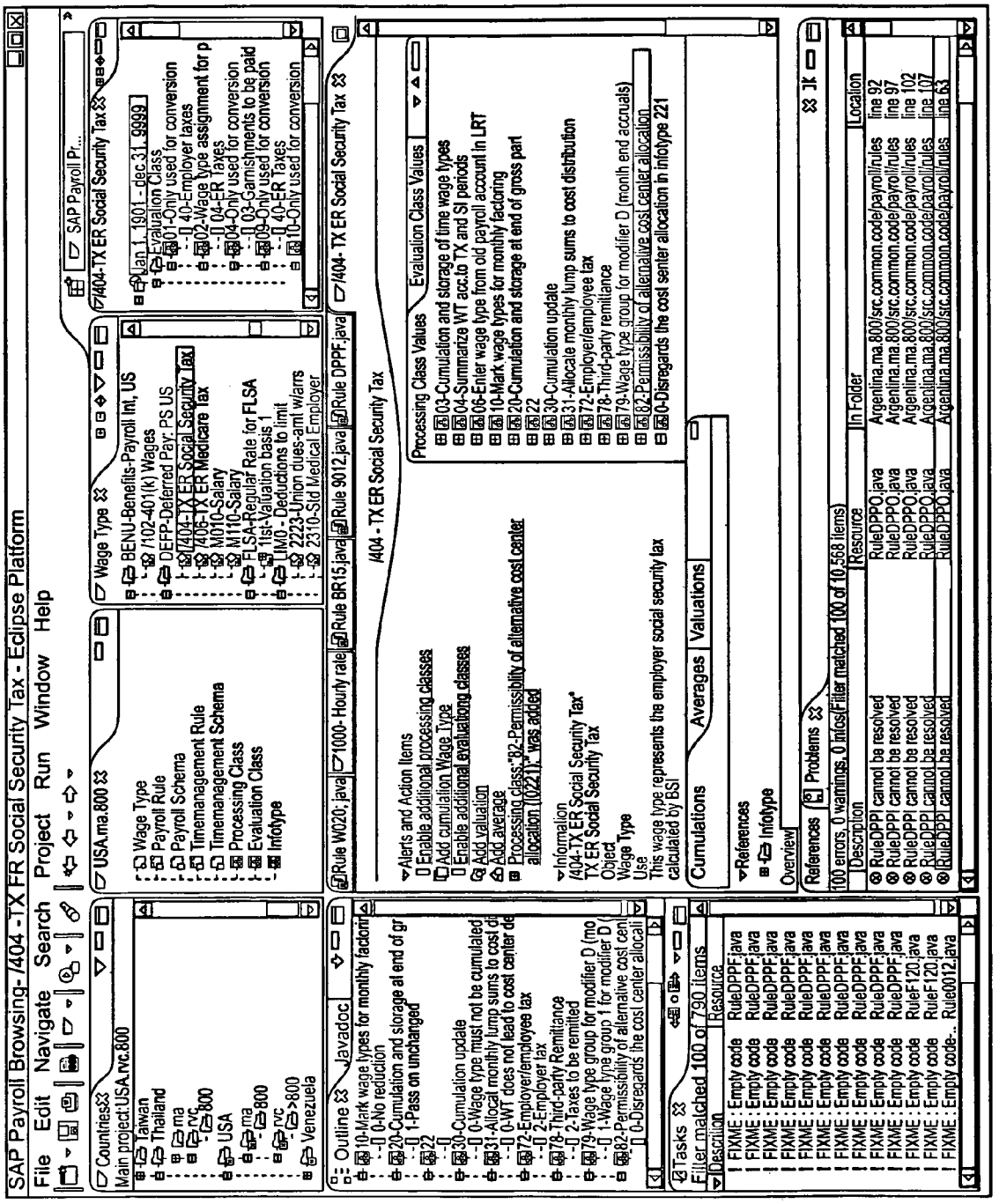
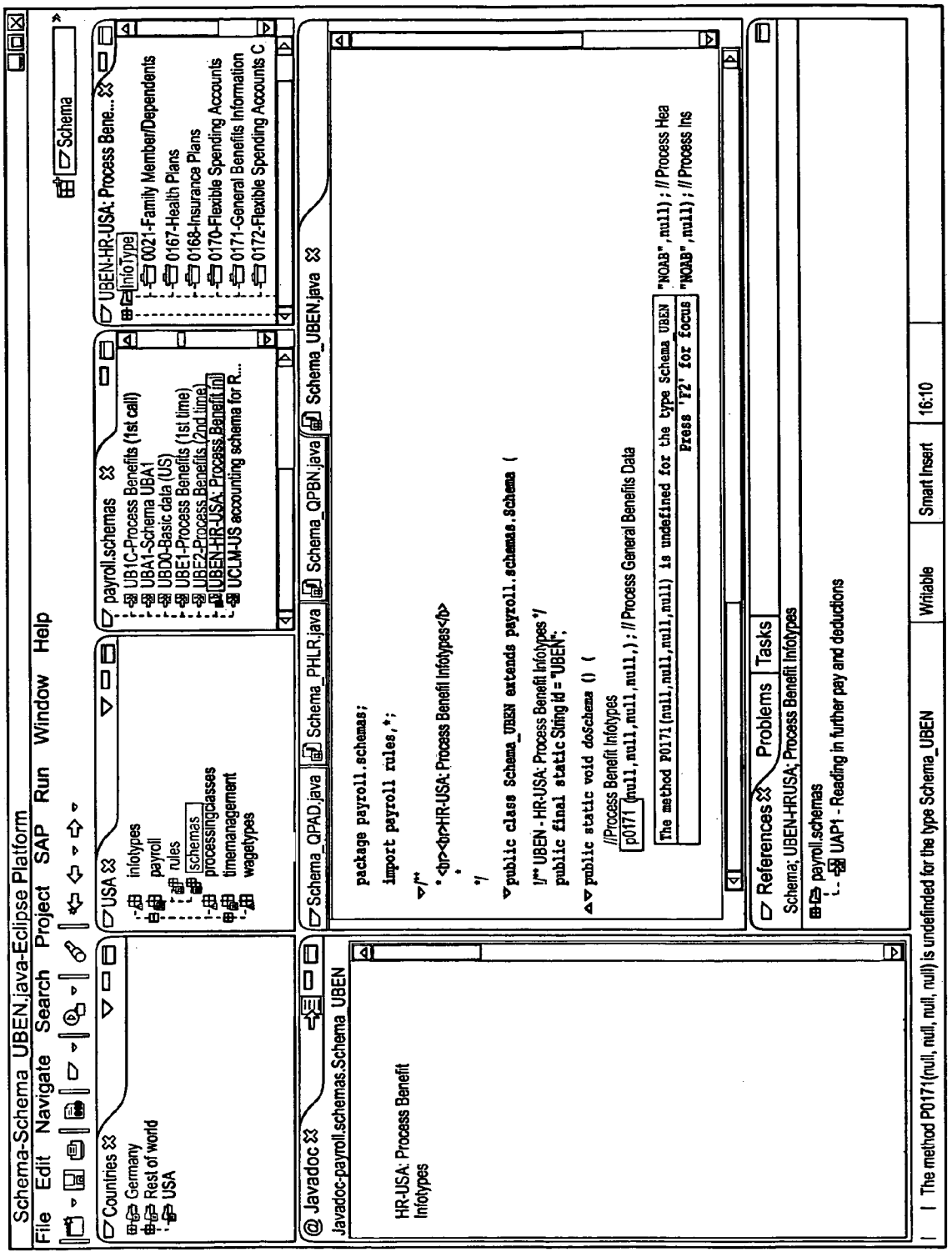

SAP Payroll Browsing- /404 -TX FR Social Security Tax - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Countries
Main project:USA.rvc.800
Taiwan
Thailand
ma
rvc
800
USA
ma
800
rvc
>800
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Wage Type
BENU-Benefits-Payroll Int, US
/102-401(k) Wages
DEFP-Deferred Pay: PS US
/404-TX ER Social Security Tax
406-TX ER Medicare tax
M010-Salary
M110-Salary
FLSA-Regular Rate for FLSA
1ist-Valuation basis 1
IM0 - Deductions to limit
2223-Union dues-amt w/arrs
2310-Std Medical Employer
/404-TX ER Social Security Tax
Jan 1, 1901 - dec 31, 9999
Evaluation Class
01-Only used for conversion
40-Employer taxes
02-Wage type assignment for p
04-ER Taxes
04-Only used for conversion
03-Garnishments to be paid
09-Only used for conversion
40-ER Taxes
10-Only used for conversion
Rule W020; java
1000- Hourly rate
Rule BR15;java
Rule 9012.java
Rule DPPF.java
/404- TX ER Social Security Tax
/404 - TX ER Social Security Tax
Alerts and Action Items
Enable additional processing classes
Add cumulation Wage Type
Enable additional evaluating classes
Add valuation
Add average
Processing class:"82-Permissibility of alternative cost center allocation (0221)" was added
Information
/404-TX ER Social Security Tax"
TX ER Social Security Tax
Object
Wage Type
Use
This wage type represents the employer social security tax calculated by BSI
Processing Class Values
Evaluation Class Values
03-Cumulation and storage of time wage types
04-Summarize WT acc.to TX and SI periods
06-Enter wage type from old payroll account in LRT
10-Mark wage types for monthly factoring
20-Cumulation and storage at end of gross part
22
30-Cumulation update
31-Allocate monthly lump sums to cost distribution
72-Employer/employee tax
78-Third-party remittance
79-Wage type group for modifier D (month end accruals)
82-Permissibility of alternative cost center allocation
0-Disregards the cost center allocation in infotype 221
Cumulations
Averages
Valuations
References
Infotype
Overview
Outline
Javadoc
10-Mark wage types for monthly factorin
0-No reduction
20-Cumulation and storage at end of gr
1-Pass on unchanged
22
30-Cumulation update
0-Wage type must not be cumulated
31-Allocat monthly lump sums to cost di
0-WT does not lead to cost center de
72-Employer/employee tax
2-Employer tax
78-Third-party Remittance
2-Taxes to be remitted
79-Wage type group for modifier D (mo
1-Wage type group 1 for modifier D (
82-Permissibility of alternative cost cen
0-Disregards the cost center allocati
Tasks
Filter matched 100 of 790 items
Description
Resource
FIXME : Empty code
RuleDPPF.java
References
Problems
100 errors, 0 warnings, 0 infos (Filter matched 100 of 10,568 items)
Description
Resource
In Folder
Location
RuleDPPI cannot be resolved
RuleDPPO.java
Argentina.ma.800/src.common.code/payroll/rules
line 92
line 97
line 102
line 107
line 63

FIG. 23


Schema-Schema_UBEN.java-Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
Schema
Countries
Germany
Rest of world
USA
USA
infotypes
payroll
rules
schemas
processingclasses
timemanagement
wagetypes
payroll.schemas
UB1C-Process Benefits (1st call)
UBA1-Schema UBA1
UBD0-Basic data (US)
UBE1-Process Benefits (1st time)
UBE2-Process Benefits (2nd time)
UBEN-HR-USA: Process Benefit Inf
UCLM-US accounting schema for R...
UBEN-HR-USA: Process Bene...
InfoType
0021-Family Member/Dependents
0167-Health Plans
0168-Insurance Plans
0170-Flexible Spending Accounts
0171-General Benefits Information
0172-Flexible Spending Accounts C
Javadoc
Javadoc-payroll.schemas.Schema_UBEN
HR-USA: Process Benefit Infotypes
Schema_QPAD.java
Schema_PHLR.java
Schema_QPBN.java
Schema_UBEN.java
package payroll.schemas;
import payroll rules,*;
/**
*

HR-USA: Process Benefit Infotypes
*
*/
public class Schema_UBEN extends payroll.schemas.Schema {
//* UBEN - HR-USA: Process Benefit Infotypes */
public final static String id = "UBEN";
public static void doSchema () {
//Process Benefit Infotypes
p0171 (null,null,null,null,null,); // Process General Benefits Data
The method P0171(null,null,null,null) is undefined for the type Schema_UBEN "NOAB",null); // Process Hea
Press 'F2' for focus "NOAB",null); // Process Ins
References
Problems
Tasks
Schema: UBEN-HRUSA: Process Benefit Infotypes
payroll.schemas
UAP1 - Reading in further pay and deductions
The method P0171(null, null, null, null) is undefinded for the type Schema_UBEN
Writable
Smart Insert
16:10

FIG. 24

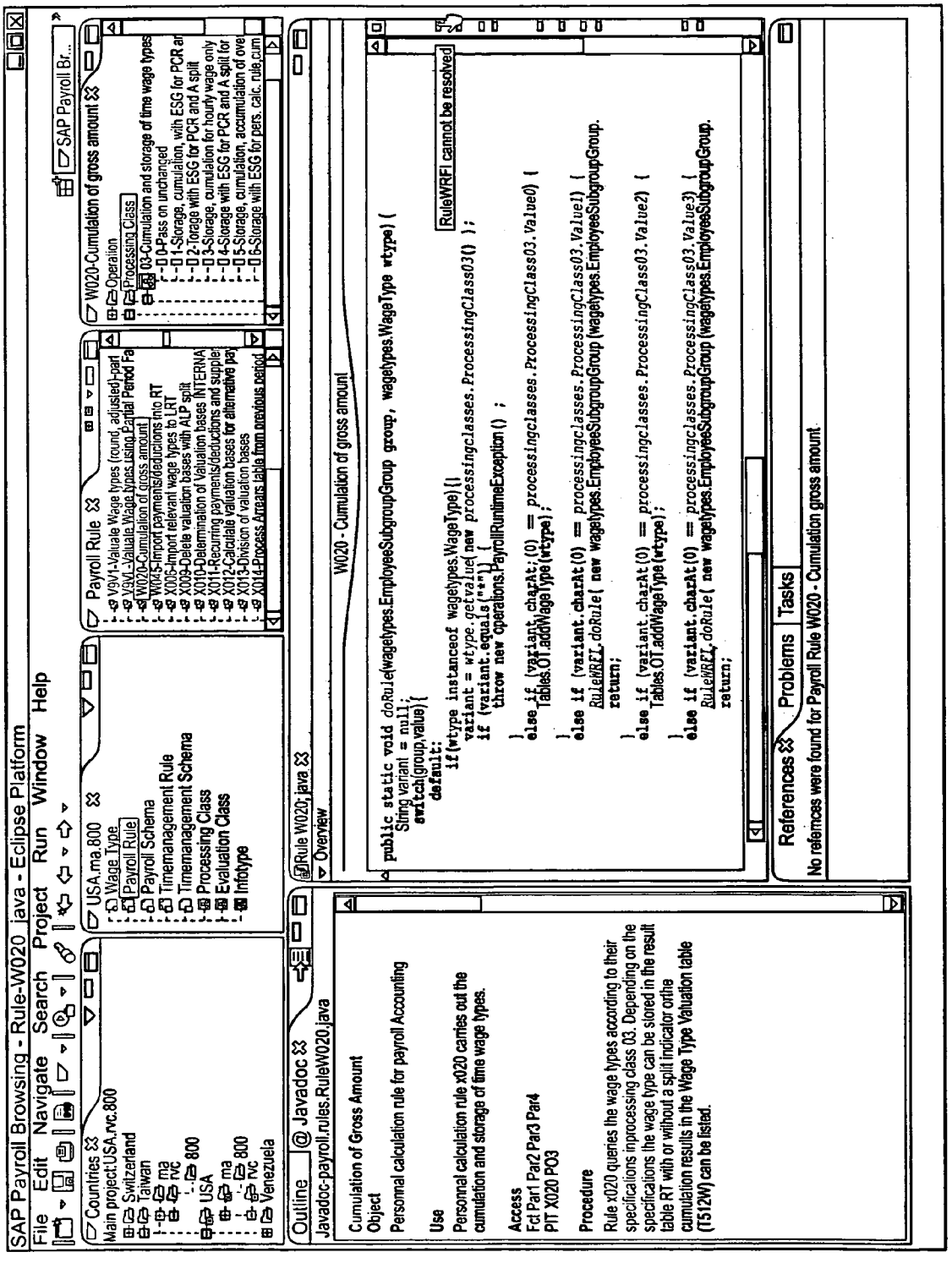

SAP Payroll Browsing - Rule-W020.java - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Countries
Main project:USA.rvc.800
Switzerland
Taiwan
ma
rvc
800
USA
ma
800
rvc
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
V9V1-Valuate Wage types (round, adjusted)-part
V9VL-Valuate Wage types using Partial Period Fa
W020-Cumulation of gross amount
W045-Import payments/deductions into RT
X008-Import relevant wage types to LRT
X009-Delete valuation bases with ALP split
X010-Determination of Valuation bases INTERNA
X011-Recurring payments/deductions and supplei
X012-Calculate valuation bases for alternative pa
X013-Division of valuation bases
X014-Process Arrears table from previous period
W020-Cumulation of gross amount
Operation
Processing Class
03-Cumulation and storage of time wage types
0-Pass on unchanged
1-Storage, cumulation, with ESG for PCR ar
2-Storage with ESG for PCR and A split
3-Storage, cumulation for hourly wage only
4-Storage with ESG for PCR and A split for
5-Storage, cumulation, accumulation of ove
6-Storage with ESG for pers. calc. rule,cum
Outline
Javadoc
Javadoc-payroll.rules.RuleW020.java
Cumulation of Gross Amount
Object
Personnal calculation rule for payroll Accounting
Use
Personnal calculation rule x020 carries out the cumulation and storage of time wage types.
Access
Fct Par1 Par2 Par3 Par4
PIT X020 P03
Procedure
Rule x020 queries the wage types according to their specifications inprocessing class 03. Depending on the specifications the wage type can be stored in the result table RT with or without a split indicator orthe cumulation results in the Wage Type Valuation table (T512W) can be listed.
RuleW020.java
Overview
W020 - Cumulation of gross amount
public static void doRule(wagetypes.EmployeeSubgroupGroup group, wagetypes.WageType wtype) {
String variant = null;
switch(group.value){
default:
if(wtype instanceof wagetypes.WageType){
variant = wtype.getValue(new processingclasses.ProcessingClass03());
if (variant.equals("**")) {
throw new operations.PayrollRuntimeException() ;
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value0) {
Tables.OT.addWageType(wtype);
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value1) {
RuleWRFI.doRule(new wagetypes.EmployeeSubgroupGroup (wagetypes.EmployeeSubgroupGroup.
return;
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value2) {
Tables.OT.addWageType(wtype);
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value3) {
RuleWRFI.doRule(new wagetypes.EmployeeSubgroupGroup (wagetypes.EmployeeSubgroupGroup.
return;
RuleWRFI cannot be resolved
References
Problems
Tasks
No references were found for Payroll Rule W020 - Cumulation gross amount.

FIG. 25

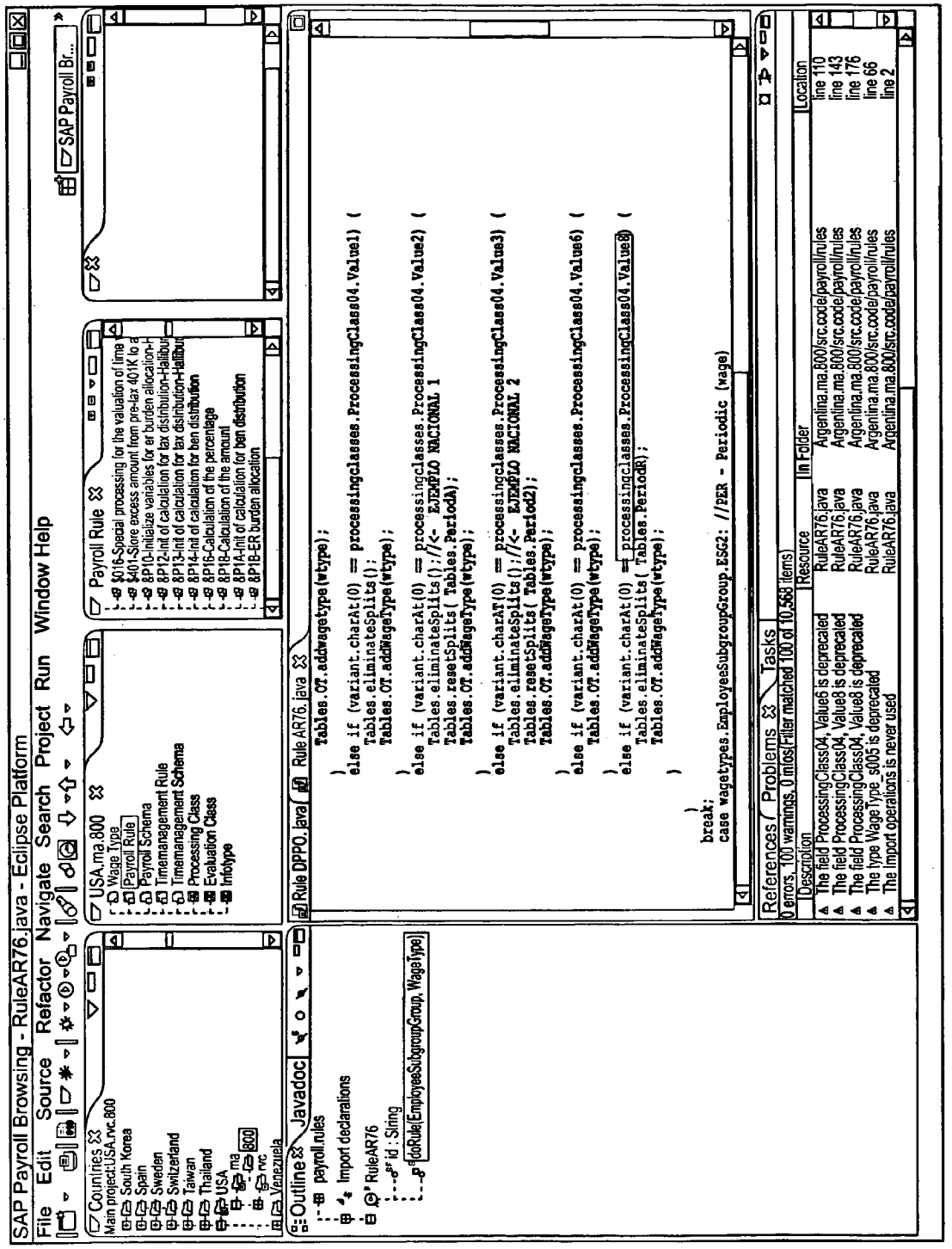

SAP Payroll Browsing - RuleAR76.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Countries
Main project:USA.rvc.800
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
USA
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
$016-Special processing for the valuation of time
$401-Store excess amount from pre-tax 401K to a
BP10-Initialize variables for er burden allocation-P
BP12-Init of calculation for tax distribution-Halibur
BP13-Init of calculation for tax distribution-Halibur
BP14-Init of calculation for ben distribution
BP16-Calculation of the percentage
BP18-Calculation of the amount
BP1A-Init of calculation for ben distribution
BP1B-ER burden allocation
SAP Payroll Br...
Outline
Javadoc
payroll.rules
Import declarations
RuleAR76
id : String
doRule(EmployeeSubgroupGroup, WageType)
Rule DPPO.java
RuleAR76.java
Tables.CT.addwagetype(wtype);
),
else if (variant.charAt(0) == processingclasses.ProcessingClass04.Value1) (
Tables.eliminateSplits();
Tables.CT.addWageType(wtype);
),
else if (variant.charAt(0) == processingclasses.ProcessingClass04.Value2) (
Tables.eliminateSplits();//<- EJEMPLO NACIONAL 1
Tables.resetSplits(Tables.Period1);
Tables.CT.addWageType(wtype);
),
else if (variant.charAT(0) == processingclasses.ProcessingClass04.Value3) (
Tables.eliminateSplits();//<- EJEMPLO NACIONAL 2
Tables.resetSplits(Tables.Period2);
Tables.CT.addWageType(wtype);
),
else if (variant.charAt(0) == processingclasses.ProcessingClass04.Value6) (
Tables.CT.addWageType(wtype);
),
else if (variant.charAt(0) == processingclasses.ProcessingClass04.Value8) (
Tables.eliminateSplits(Tables.Period8);
Tables.CT.addWageType(wtype);
)
)
break;
case wagetypes.EmployeeSubgroupGroup.ES62: //PER - Periodic (wage)
References
Problems
Tasks
0 errors, 100 warnings, 0 infos (Filter matched 100 of 10,568 items)
Description
Resource
In Folder
Location
The field ProcessingClass04, Value6 is deprecated
The field ProcessingClass04, Value8 is deprecated
The field ProcessingClass04, Value8 is deprecated
The type WageType_s005 is deprecated
The import operations is never used
RuleAR76.java
Argentina.ma.800/src.code/payroll/rules
line 110
line 143
line 176
line 66
line 2

FIG. 26

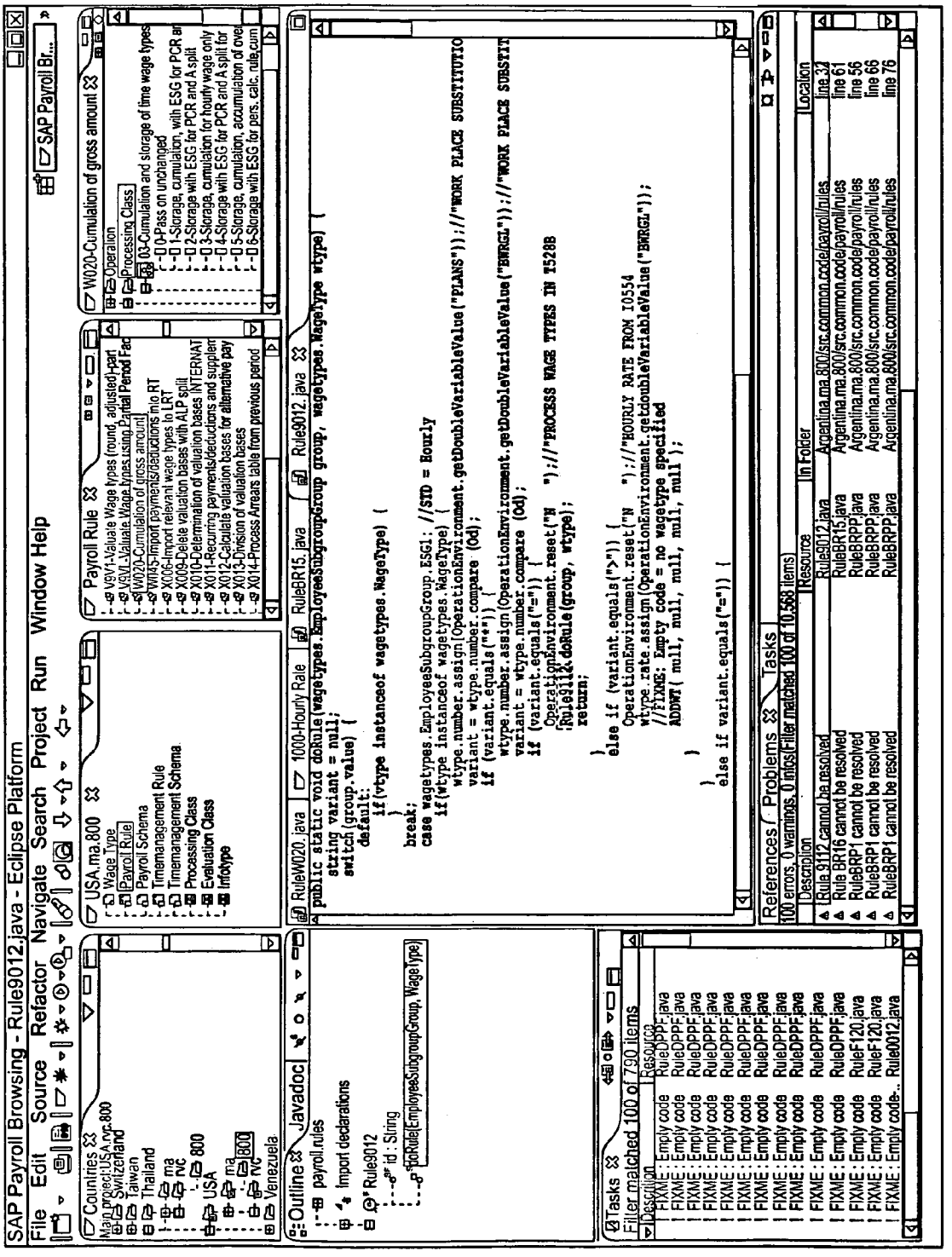

SAP Payroll Browsing - Rule9012.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Countries
Main project:USA nyc.800
Switzerland
Taiwan
Thailand
ma
nyc
800
USA
ma
nyc
800
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
V6V1-Valuate Wage types (round, adjusted)-part
V6VL-Valuate Wage types using Partial Period Fac
W020-Cumulation of gross amount
W145-Import payments/deductions into RT
X006-Import relevant wage types to LRT
X009-Delete valuation bases with ALP split
X010-Determination of valuation bases INTERNAT
X011-Recurring payments/deductions and supplem
X012-Calculate valuation bases for alternative pay
X013-Division of valuation bases
X014-Process Arrears table from previous period
W020-Cumulation of gross amount
Operation
Processing Class
03-Cumulation and storage of time wage types
0-Pass on unchanged
1-Storage, cumulation, with ESG for PCR an
2-Storage with ESG for PCR and A split
3-Storage, cumulation for hourly wage only
4-Storage with ESG for PCR and A split for
5-Storage, cumulation, accumulation of over
6-Storage with ESG for pers. calc. rule.cum
Outline
Javadoc
payroll.rules
Import declarations
Rule9012
id : String
doRule(EmployeesSubgroupGroup, WageType)
RuleW020.java
1000-Hourly Rule
RuleBR15.java
Rule9012.java
public static void doRule(wagetypes.EmployeesSubgroupGroup group, wagetypes.WageType wtype) {
string variant = null;
switch (group.value) {
default:
if(wtype instanceof wagetypes.WageType) {
}
break;
case wagetypes.EmployeesSubgroupGroup.E561: //STD = Hourly
if(wtype instanceof wagetypes.WageType) {
wtype.number.assign(OperationEnvironment.getDoubleVariableValue("PLANS"));//"WORK PLACE SUBSTITUTIO
variant = wtype.number.compare (0d);
if (variant.equals("=")) {
wtype.number.assign(OperationEnvironment.getDoubleVariableValue("ENRGL"));//"WORK PLACE SUBSTIT
variant = wtype.number.compare (0d);
if (variant.equals("=")) {
OperationEnvironment.reset("N ");//"PROCESS WAGE TYPES IN T528B
Rule9112.doRule(group, wtype);
return;
}
else if (variant.equals(">")) {
OperationEnvironment.reset("N ");//"HOURLY RATE FROM I0554
wtype.rate.assign(OperationEnvironment.getDoubleVariableValue("ENRGL"));
//FIXME: Empty code = no wagetype specified
ADDWT(null, null, null, null, null);
}
}
else if (variant.equals("=")) {
Tasks
Filter matched 100 of 790 items
Description
Resource
FIXME : Empty code RuleDPPF.java
References
Problems
Tasks
100 errors, 0 warnings, 0 infos (Filter matched 100 of 10,568 items)
Description
Resource
In Folder
Location
Rule 9112 cannot be resolved Rule9012.java Argentina.ma.800/src.common.code/payroll/rules line 32
Rule BR16 cannot be resolved RuleBR15.java Argentina.ma.800/src.common.code/payroll/rules line 61
RuleBRP1 cannot be resolved RuleBRPP.java Argentina.ma.800/src.common.code/payroll/rules line 56
RuleBRP1 cannot be resolved RuleBRPP.java Argentina.ma.800/src.common.code/payroll/rules line 66
RuleBRP1 cannot be resolved RuleBRPP.java Argentina.ma.800/src.common.code/payroll/rules line 76

FIG. 27

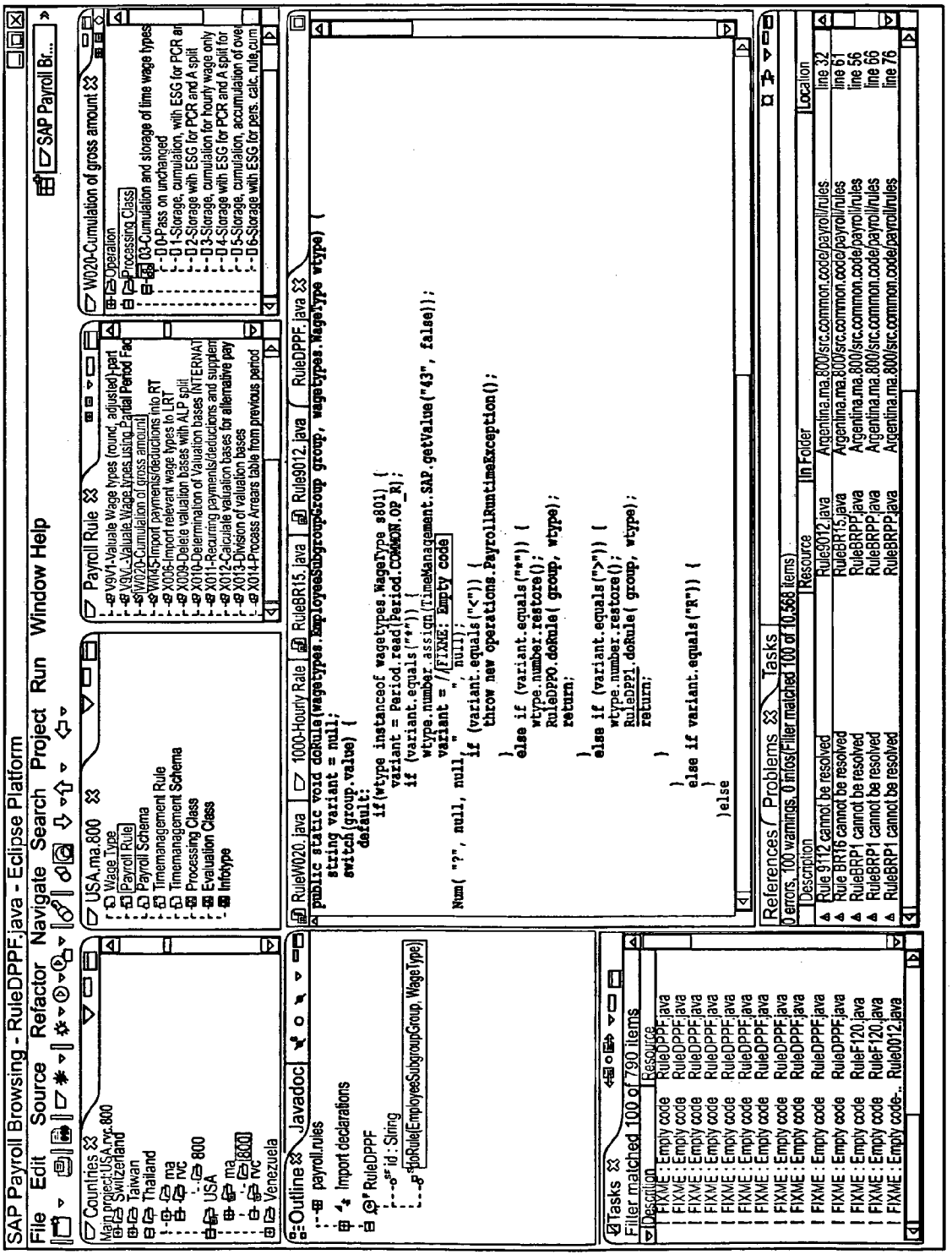

SAP Payroll Browsing - RuleDPPF.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
SAP Payroll Br...
Countries
Main project:USA.rvc.800
Switzerland
Taiwan
Thailand
ma
rvc
800
USA
ma
800
rvc
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
V9U1-Valuate Wage types (round, adjusted) part
V9V-Valuate Wage types using Partial Period Fac
W020-Cumulation of gross amount
W045-Import payments/deductions into RT
X006-Import relevant wage types to LRT
X009-Delete valuation bases with ALP split
X010-Determination of Valuation bases INTERNAT
X011-Recurring payments/deductions and supplem
X012-Calculate valuation bases for alternative pay
X013-Division of valuation bases
X014-Process Arrears table from previous period
W020-Cumulation of gross amount
Operation
Processing Class
03-Cumulation and storage of time wage types
10-Pass on unchanged
11-Storage, cumulation, with ESG for PCR ar
12-Storage with ESG for PCR and A split
13-Storage, cumulation for hourly wage only
14-Storage with ESG for PCR and A split for
15-Storage, cumulation, accumulation of over
16-Storage with ESG for pers. calc. rule,cum
Outline
Javadoc
payroll.rules
Import declarations
RuleDPPF
id : String
doRule(EmployeesSubgroupGroup, WageType)
RuleW020.java
1000-Hourly Rate
RuleBR15.java
Rule9012.java
RuleDPPF.java
public static void doRule(wagetypes.EmployeesSubgroupGroup group, wagetypes.WageType wtype) {
string variant = null;
switch(group.value) {
default:
if(wtype instanceof wagetypes.WageType_s801) {
variant = Period.read(Period.COMMON_02_R);
if (variant.equals("*")) {
wtype.number.assign(TimeManagement.SAP.getValue("43", false));
variant = /*FIXME: Empty code
Num("?", null, null, "
, null);
if (variant.equals("<")) {
throw new operations.PayrollRuntimeException();
}
else if (variant.equals("*")) {
wtype.number.restore();
RuleDPP0.doRule(group, wtype);
return;
}
else if (variant.equals(">")) {
wtype.number.restore();
RuleDPP1.doRule(group, wtype);
return;
}
else if (variant.equals("R")) {
}
}else
Tasks
Filter matched 100 of 790 items
Description
Resource
FIXME : Empty code
RuleDPPF.java
References
Problems
Tasks
0 errors, 100 warnings, 0 infos (Filter matched 100 of 10,568 items)
Description
Resource
In Folder
Location
Rule 9112 cannot be resolved
Rule9012.java
Argentina.ma.800/src.common.code/payroll/rules
line 32
Rule BR16 cannot be resolved
RuleBR15.java
Argentina.ma.800/src.common.code/payroll/rules
line 61
RuleBRP1 cannot be resolved
RuleBRPP.java
Argentina.ma.800/src.common.code/payroll/rules
line 56
RuleBRP1 cannot be resolved
RuleBRPP.java
Argentina.ma.800/src.common.code/payroll/rules
line 66
RuleBRP1 cannot be resolved
RuleBRPP.java
Argentina.ma.800/src.common.code/payroll/rules
line 76

FIG. 28

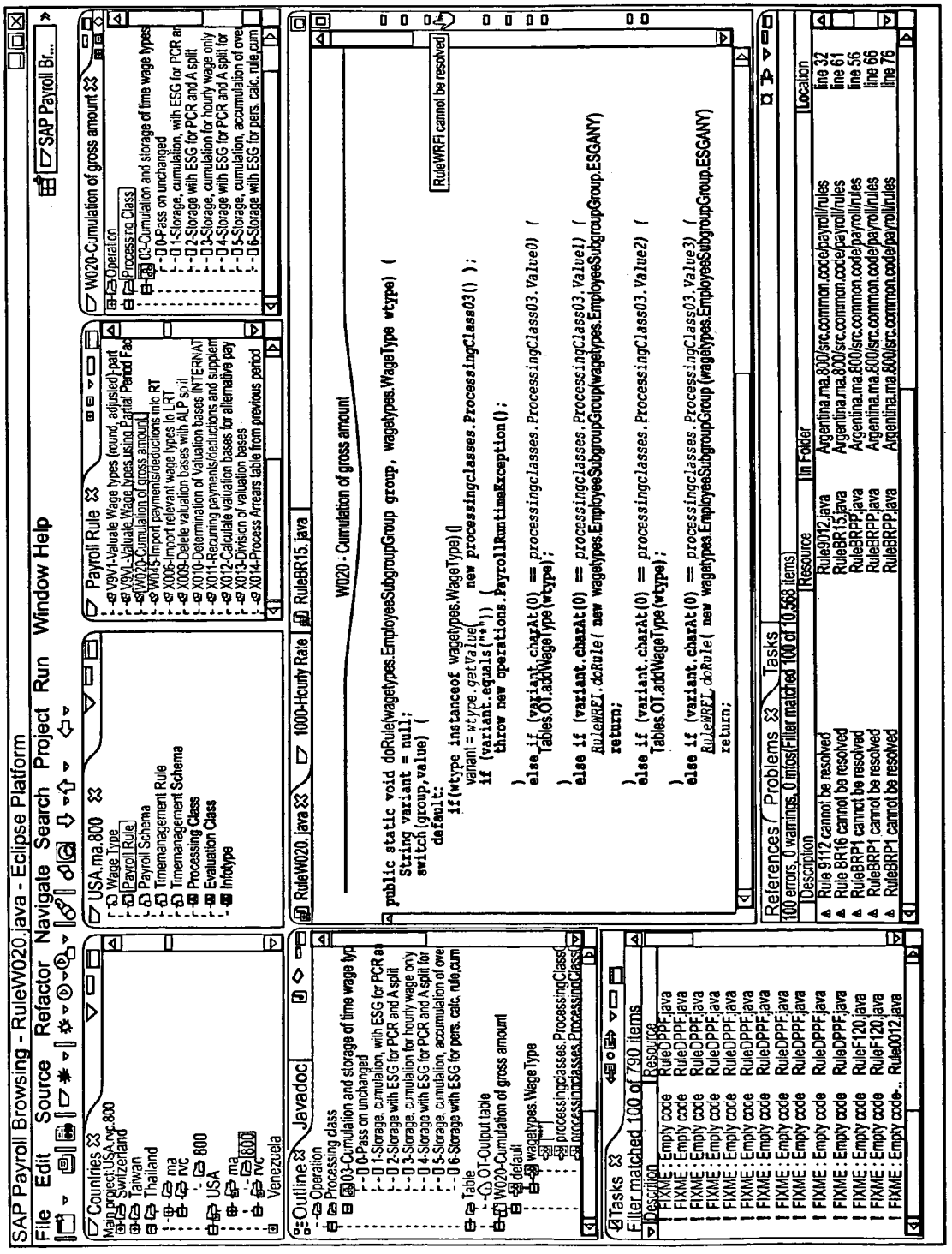

SAP Payroll Browsing - RuleW020.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
SAP Payroll Br...
Countries
Main project:USA.rvc.800
Switzerland
Taiwan
Thailand
ma
rvc
800
USA
ma
800
rvc
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
V9V1-Valuate Wage types (round, adjusted) part
V9V1_Valuate Wage types using Partial Period Fac
W020-Cumulation of gross amount
W045-Import payments/deductions into RT
X006-Import relevant wage types to LRT
X009-Delete valuation bases with ALP split
X010-Determination of Valuation bases INTERNAT
X011-Recurring payments/deductions and supplem
X012-Calculate valuation bases for alternative pay
X013-Division of valuation bases
X014-Process Arrears table from previous period
W020-Cumulation of gross amount
Operation
Processing Class
03-Cumulation and storage of time wage types
0-Pass on unchanged
1-Storage, cumulation, with ESG for PCR ar
2-Storage with ESG for PCR and A split
3-Storage, cumulation for hourly wage only
4-Storage with ESG for PCR and A split for
5-Storage, cumulation, accumulation of ove
6-Storage with ESG for pers. calc. rule.cum
Outline
Javadoc
Operation
Processing class
03-Cumulation and storage of time wage typ
0-Pass on unchanged
1-Storage, cumulation, with ESG for PCR a
2-Storage with ESG for PCR and A split
3-Storage, cumulation for hourly wage only
4-Storage with ESG for PCR and A split for
5-Storage, cumulation, accumulation of ove
6-Storage with ESG for pers. calc. rule.cum
Table
OT-Output table
W020-Cumulation of gross amount
default
wagetypes.WageType
processingclasses.ProcessingClass
processingclasses.ProcessingClass
RuleW020.java
1000-Hourly Rate
RuleBR15.java
W020 - Cumulation of gross amount
public static void doRule(wagetypes.EmployeeSubgroupGroup group, wagetypes.WageType wtype) {
String variant = null;
switch (group.value) {
default:
if(wtype instanceof wagetypes.WageType){
variant = wtype.getValue();
if (variant.equals("*")) {
throw new operations.PayrollRuntimeException();
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value0) {
Tables.OT.addWageType(wtype);
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value1) {
RuleWRFI.doRule(new wagetypes.EmployeeSubgroupGroup(wagetypes.EmployeeSubgroupGroup.ESGANY)
return;
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value2) {
Tables.OT.addWageType(wtype);
}
else if (variant.charAt(0) == processingclasses.ProcessingClass03.Value3) {
RuleWRFI.doRule(new wagetypes.EmployeeSubgroupGroup (wagetypes.EmployeeSubgroupGroup.ESGANY)
return;
RuleWRFI cannot be resolved
Tasks
Filter matched 100 of 790 items
Description
Resource
FIXME : Empty code RuleDPPF.java
References
Problems
Tasks
100 errors, 0 warnings, 0 infos (Filter matched 100 of 10,568 items)
Description
Resource
In Folder
Location
Rule 9112 cannot be resolved
Rule BR16 cannot be resolved
RuleBRP1 cannot be resolved
RuleBRP1 cannot be resolved
RuleBRP1 cannot be resolved
Rule9012.java
RuleBR15.java
RuleBRPP.java
RuleBRPP.java
RuleBRPP.java
Argentina.ma.800/src.common.code/payroll/rules
Argentina.ma.800/src.common.code/payroll/rules
Argentina.ma.800/src.common.code/payroll/rules
Argentina.ma.800/src.common.code/payroll/rules
Argentina.ma.800/src.common.code/payroll/rules
line 32
line 61
line 56
line 66
line 76

FIG. 29

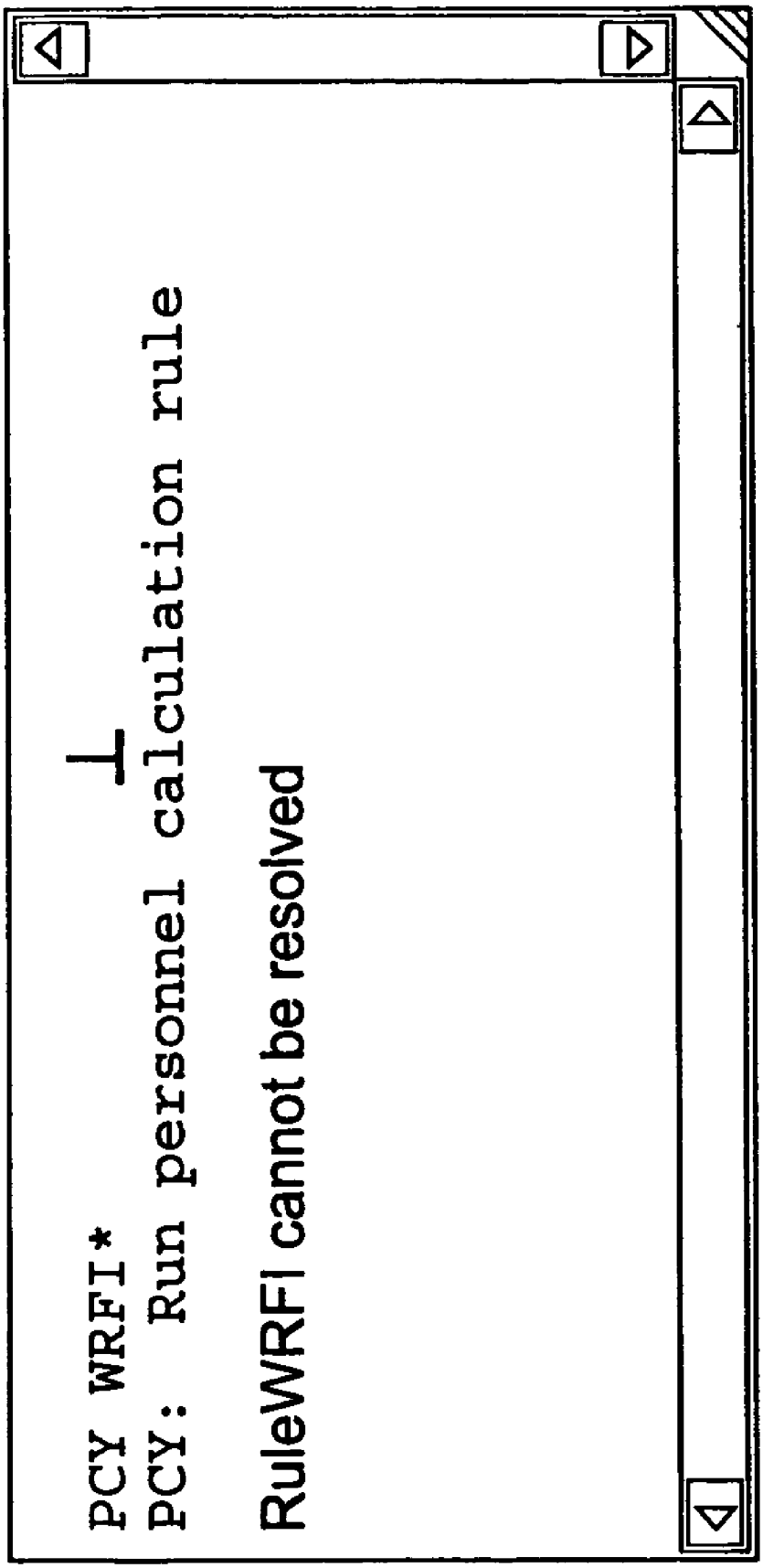

PCY WRFI*
PCY: Run personnel calculation rule
RuleWRFI cannot be resolved

FIG. 30

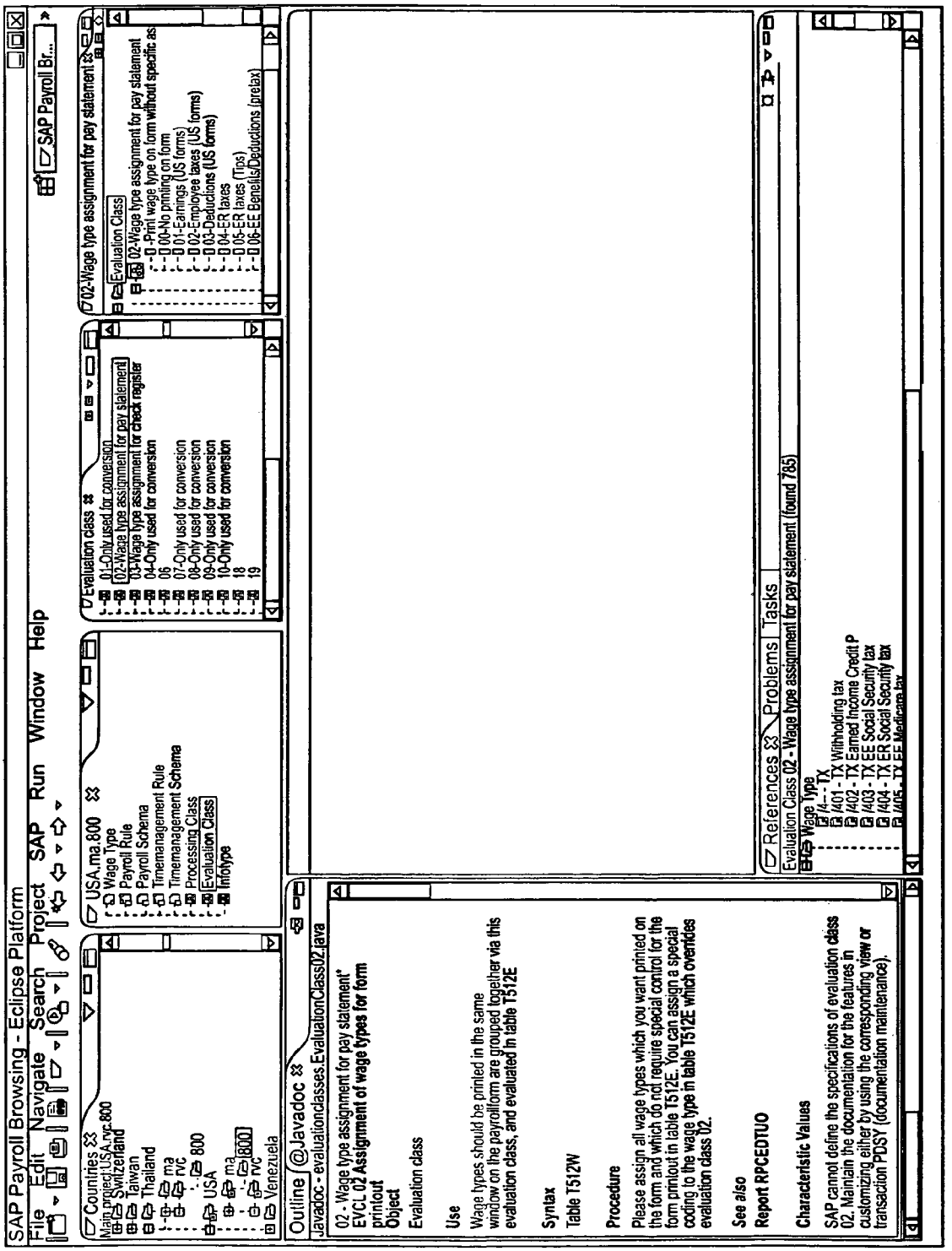

SAP Payroll Browsing - Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
SAP Payroll Br...
Countries
Main project USA nyc 800
Switzerland
Taiwan
Thailand
ma
nyc
800
USA
ma
800
nyc
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Evaluation class
01-Only used for conversion
02-Wage type assignment for pay statement
03-Wage type assignment for check register
04-Only used for conversion
06
07-Only used for conversion
08-Only used for conversion
09-Only used for conversion
10-Only used for conversion
18
19
02-Wage type assignment for pay statement
Evaluation Class
02-Wage type assignment for pay statement
0-Print wage type on form without specific as
00-No printing on form
01-Earnings (US forms)
02-Employee taxes (US forms)
03-Deductions (US forms)
04-ER taxes
05-ER taxes (Tips)
06-EE Benefits/Deductions (prelax)
Outline
@Javadoc
Javadoc - evaluationclasses.EvaluationClass02.java
02 - Wage type assignment for pay statement'
EVCL 02 Assignment of wage types for form printout
Object
Evaluation class
Use
Wage types should be printed in the same window on the payrollform are grouped togeither via this evaluation class, and evaluated in table T512E
Syntax
Table T512W
Procedure
Please assign all wage types which you want printed on the form and which do not require special control for the form printout in table T512E. You can assign a special coding to the wage type in table T512E which overrides evaluation class 02.
See also
Report RPCEDTUO
Characteristic Values
SAP cannot define the specifications of evaluation class 02. Maintain the documentation for the features in customizing either by using the corresponding view or transaction PDSY (documentation maintenance).
References
Problems
Tasks
Evaluation Class 02 - Wage type assignment for pay statement (found 785)
Wage Type
/4-- TX
/401 - TX Withholding tax
/402 - TX Earned Income Credit P
/403 - TX EE Social Security tax
/404 - TX ER Social Security tax
/405 - TX EE Medicare tax

FIG. 31

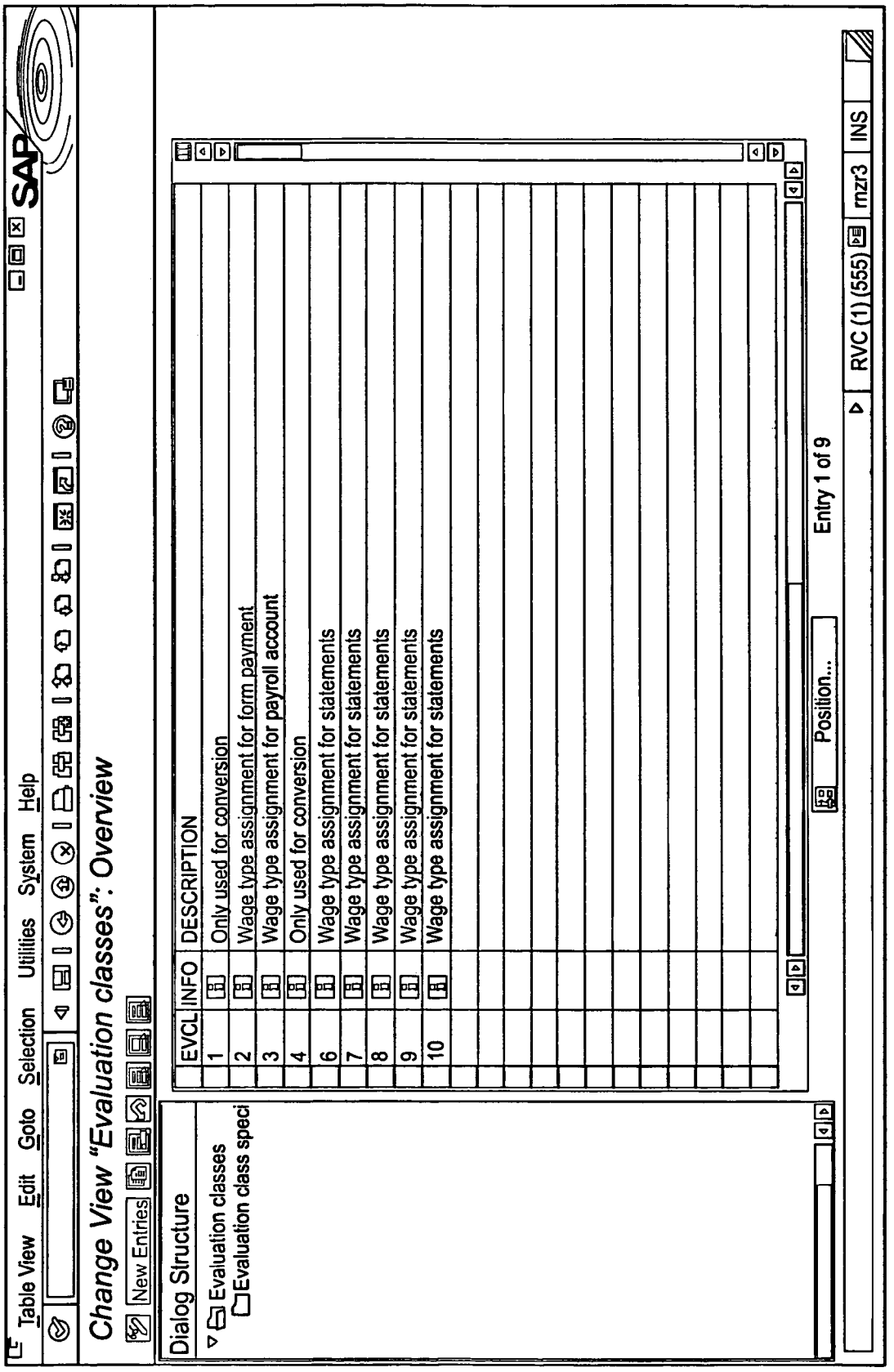


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Evaluation classes": Overview
New Entries
Dialog Structure
Evaluation classes
Evaluation class speci
EVCL INFO DESCRIPTION
1 Only used for conversion
2 Wage type assignment for form payment
3 Wage type assignment for payroll account
4 Only used for conversion
6 Wage type assignment for statements
7 Wage type assignment for statements
8 Wage type assignment for statements
9 Wage type assignment for statements
10 Wage type assignment for statements
Position...
Entry 1 of 9
RVC (1) (555)
mzr3
INS

FIG. 32

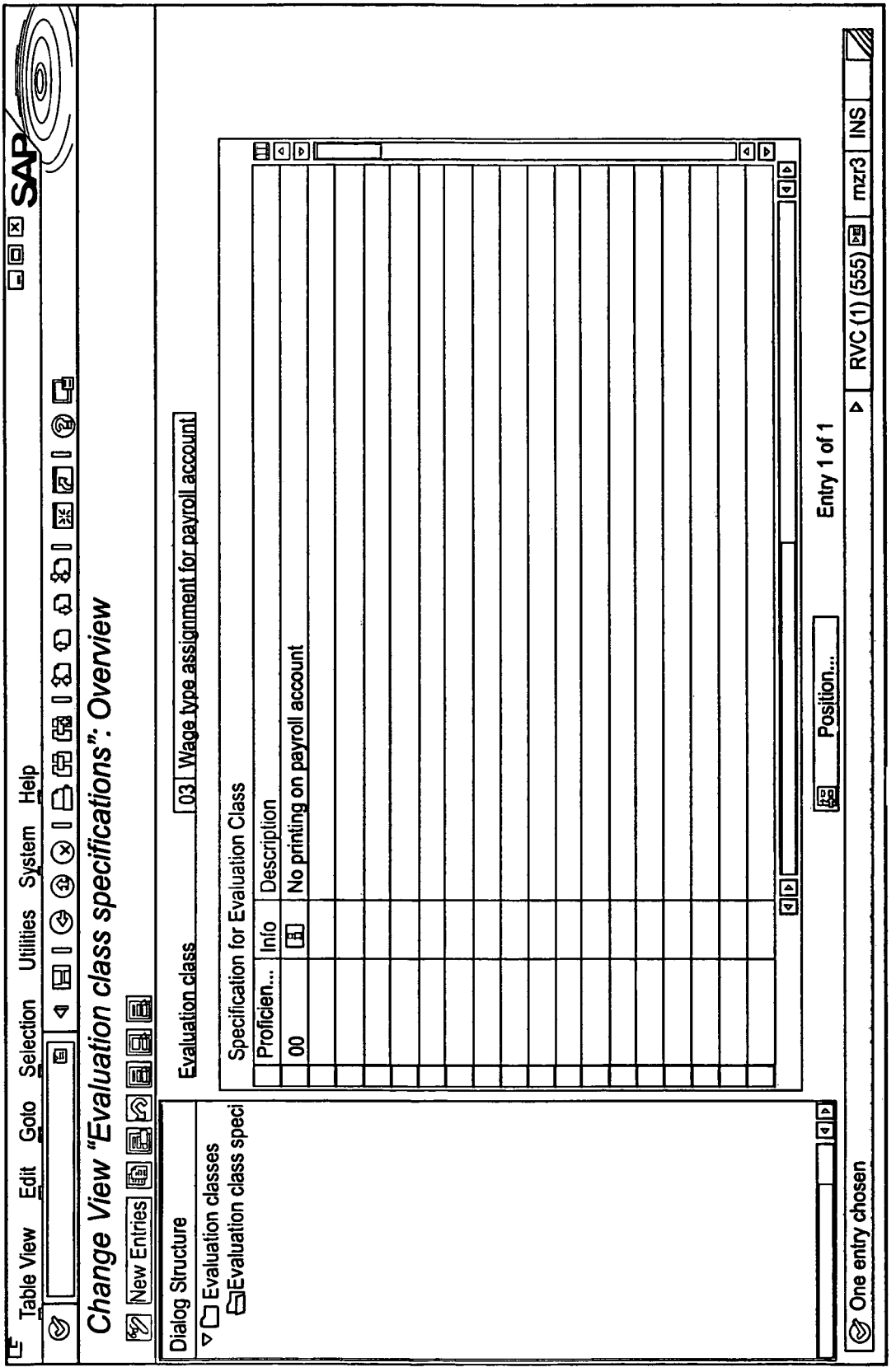


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Evaluation class specifications": Overview
New Entries
Dialog Structure
Evaluation classes
Evaluation class speci
Evaluation class
03 Wage type assignment for payroll account
Specification for Evaluation Class
Proficien...
Info
Description
00
No printing on payroll account
Position...
Entry 1 of 1
One entry chosen
RVC (1) (555)
mzr3
INS

FIG. 33

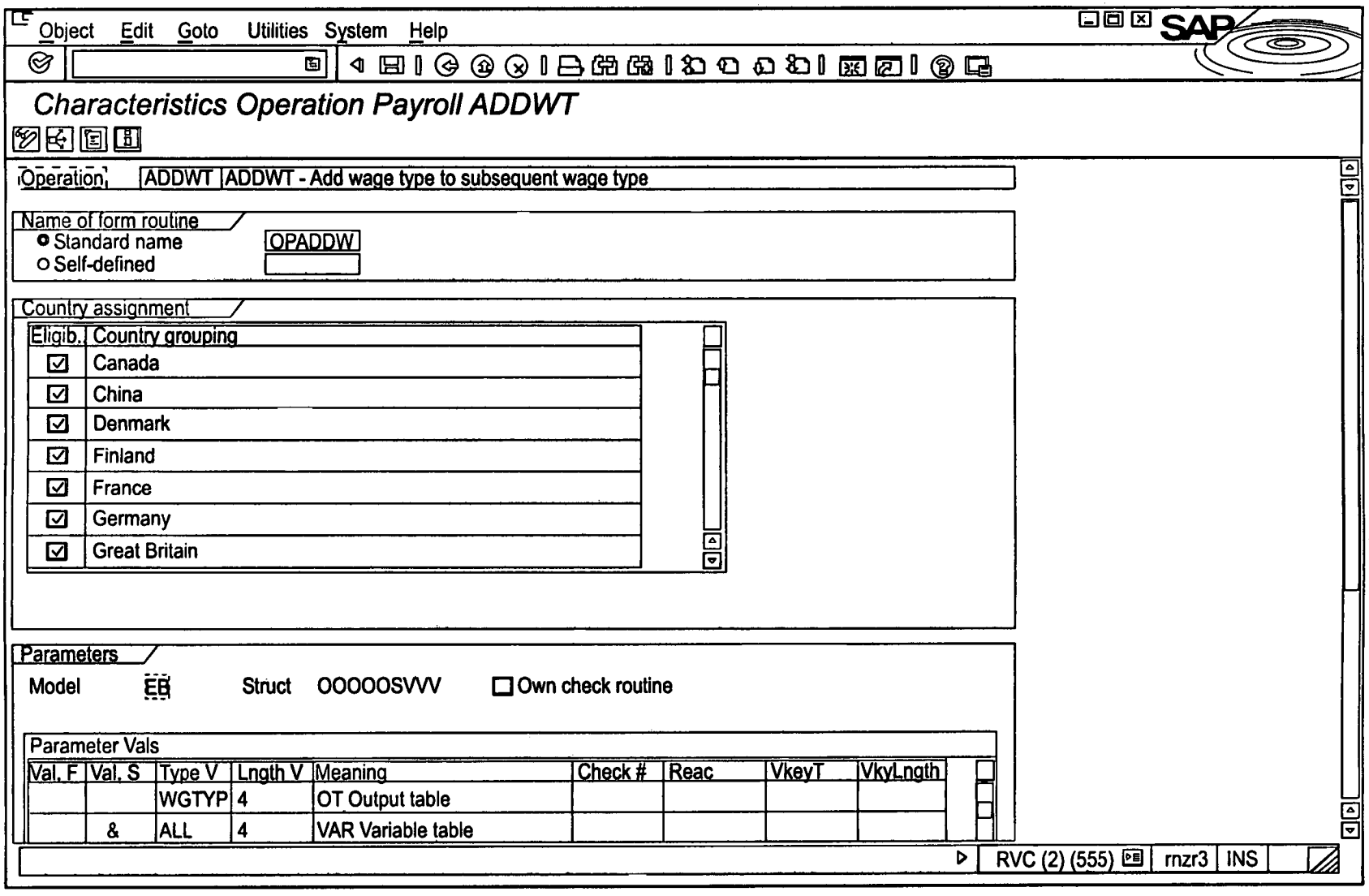

Object Edit Goto Utilities System Help
SAP
Characteristics Operation Payroll ADDWT
Operation ADDWT ADDWT - Add wage type to subsequent wage type
Name of form routine
Standard name OPADDW
Self-defined
Country assignment
Eligib. Country grouping
Canada
China
Denmark
Finland
France
Germany
Great Britain
Parameters
Model EB Struct OOOOOSVVV Own check routine
Parameter Vals
Val. F Val. S Type V Lngth V Meaning Check # Reac VkeyT VkyLngth
WGTYP 4 OT Output table
& ALL 4 VAR Variable table
RVC (2) (555) rnzr3 INS

FIG. 34

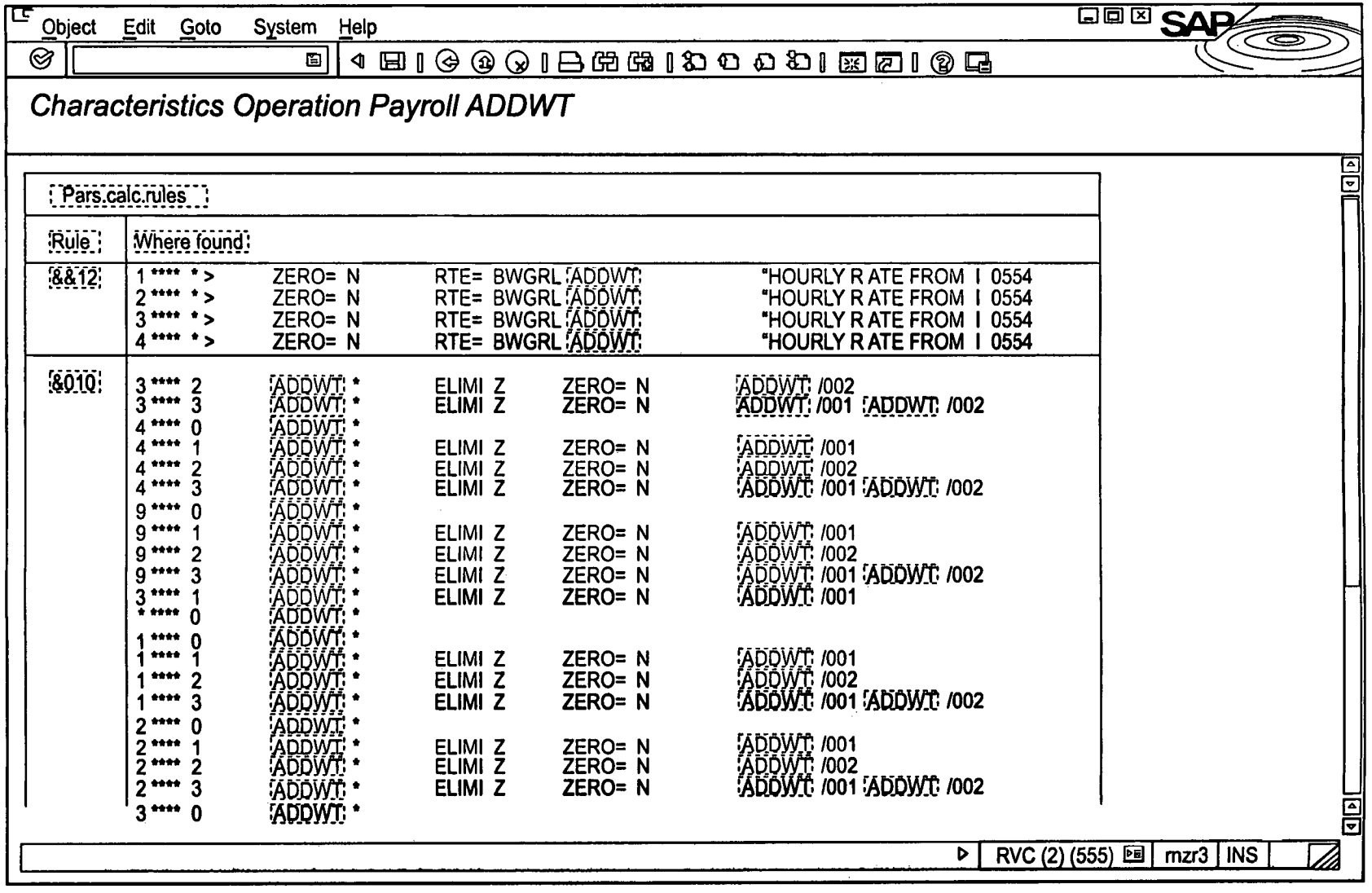

Object Edit Goto System Help
SAP
Characteristics Operation Payroll ADDWT
Pars.calc.rules
Rule
Where found
&&12
1 **** * > ZERO= N RTE= BWGRL ADDWT "HOURLY R ATE FROM I 0554
2 **** * > ZERO= N RTE= BWGRL ADDWT "HOURLY R ATE FROM I 0554
3 **** * > ZERO= N RTE= BWGRL ADDWT "HOURLY R ATE FROM I 0554
4 **** * > ZERO= N RTE= BWGRL ADDWT "HOURLY R ATE FROM I 0554
&010
3 **** 2 ADDWT * ELIMI Z ZERO= N ADDWT /002
3 **** 3 ADDWT * ELIMI Z ZERO= N ADDWT /001 ADDWT /002
4 **** 0 ADDWT *
4 **** 1 ADDWT * ELIMI Z ZERO= N ADDWT /001
4 **** 2 ADDWT * ELIMI Z ZERO= N ADDWT /002
4 **** 3 ADDWT * ELIMI Z ZERO= N ADDWT /001 ADDWT /002
9 **** 0 ADDWT *
9 **** 1 ADDWT * ELIMI Z ZERO= N ADDWT /001
9 **** 2 ADDWT * ELIMI Z ZERO= N ADDWT /002
9 **** 3 ADDWT * ELIMI Z ZERO= N ADDWT /001 ADDWT /002
3 **** 1 ADDWT * ELIMI Z ZERO= N ADDWT /001
* **** 0 ADDWT *
1 **** 0 ADDWT *
1 **** 1 ADDWT * ELIMI Z ZERO= N ADDWT /001
1 **** 2 ADDWT * ELIMI Z ZERO= N ADDWT /002
1 **** 3 ADDWT * ELIMI Z ZERO= N ADDWT /001 ADDWT /002
2 **** 0 ADDWT *
2 **** 1 ADDWT * ELIMI Z ZERO= N ADDWT /001
2 **** 2 ADDWT * ELIMI Z ZERO= N ADDWT /002
2 **** 3 ADDWT * ELIMI Z ZERO= N ADDWT /001 ADDWT /002
3 **** 0 ADDWT *
RVC (2) (555)
mzr3
INS

FIG. 35

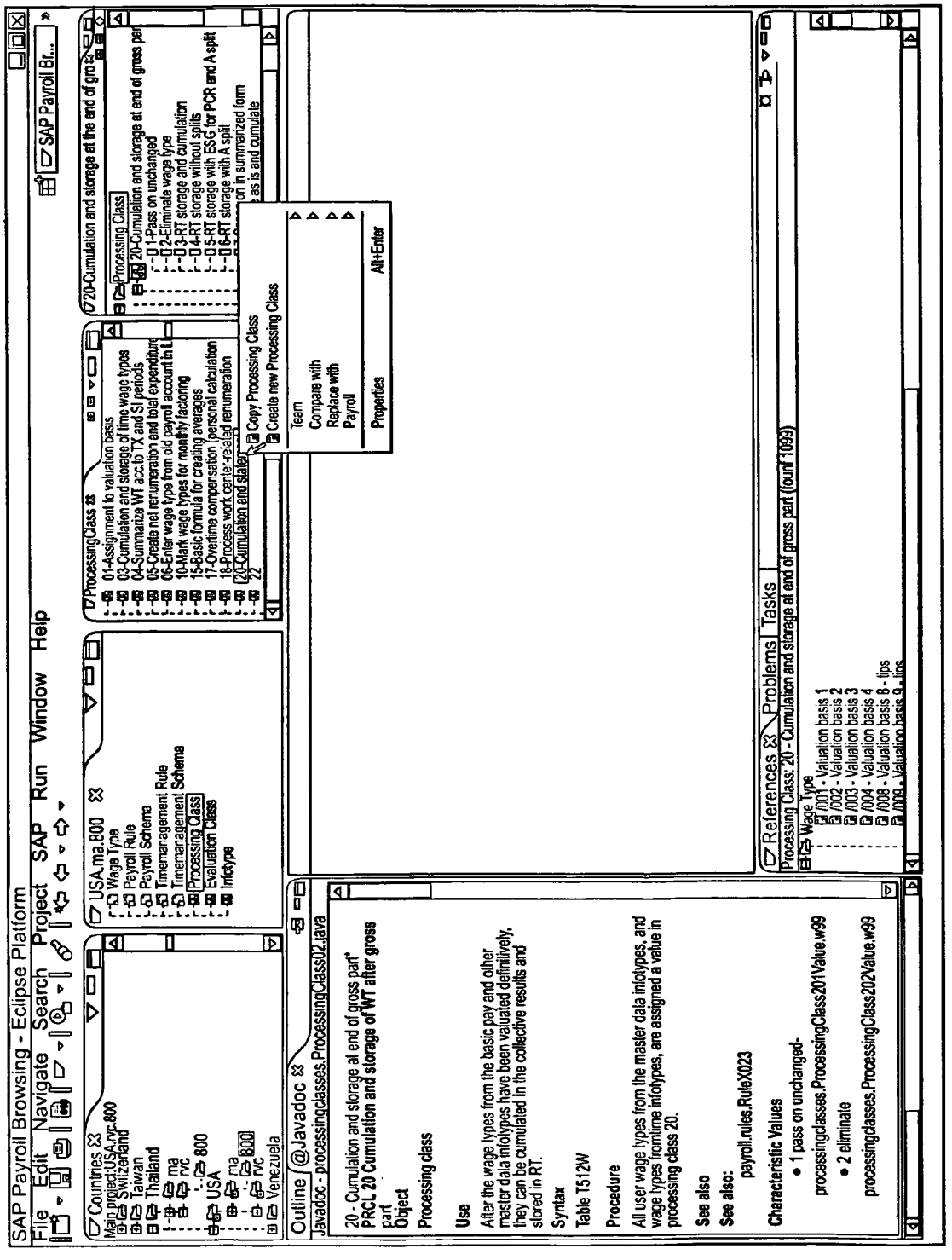

SAP Payroll Browsing - Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
SAP Payroll Br...
Countries
Main project:USA nvc.800
Switzerland
Taiwan
Thailand
ma
nvc
USA
800
ma
800
nvc
Venezuela
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
ProcessingClass
01-Assignment to valuation basis
03-Cumulation and storage of time wage types
04-Summarize WT acc.to TX and SI periods
05-Create net remuneration and total expenditure
06-Enter wage type from old payroll account in L
10-Mark wage types for monthly factoring
15-Basic formula for creating averages
17-Overtime compensation (personal calculation
18-Process work center-related remuneration
20-Cumulation and storage
22
Copy Processing Class
Create new Processing Class
20-Cumulation and storage at the end of gro
Processing Class
20-Cumulation and storage at end of gross par
1-Pass on unchanged
2-Eliminate wage type
3-RT storage and cumulation
4-RT storage without splits
5-RT storage with ESG for PCR and A split
6-RT storage with A split
as is and cumulate
Team
Compare with
Replace with
Payroll
Properties Alt+Enter
Outline
@Javadoc
Javadoc - processingclasses.ProcessingClass02.java
20 - Cumulation and storage at end of gross part"
PRCL 20 Cumulation and storage of WT after gross part
Object
Processing class
Use
After the wage types from the basic pay and other master data infotypes have been valuated definitively, they can be cumulated in the collective results and stored in RT.
Syntax
Table T512W
Procedure
All user wage types from the master data infotypes, and wage types fromtime infotypes, are assigned a value in processing class 20.
See also
See also:
payroll.rules.RuleX023
Characteristic Values
• 1 pass on unchanged-
processingclasses.ProcessingClass201Value.w99
• 2 eliminate
processingclasses.ProcessingClass202Value.w99
References
Problems
Tasks
Processing Class: 20 - Cumulation and storage at end of gross part (count 1099)
Wage Type
/001 - Valuation basis 1
/002 - Valuation basis 2
/003 - Valuation basis 3
/004 - Valuation basis 4
/008 - Valuation basis 8 - tips

FIG. 36

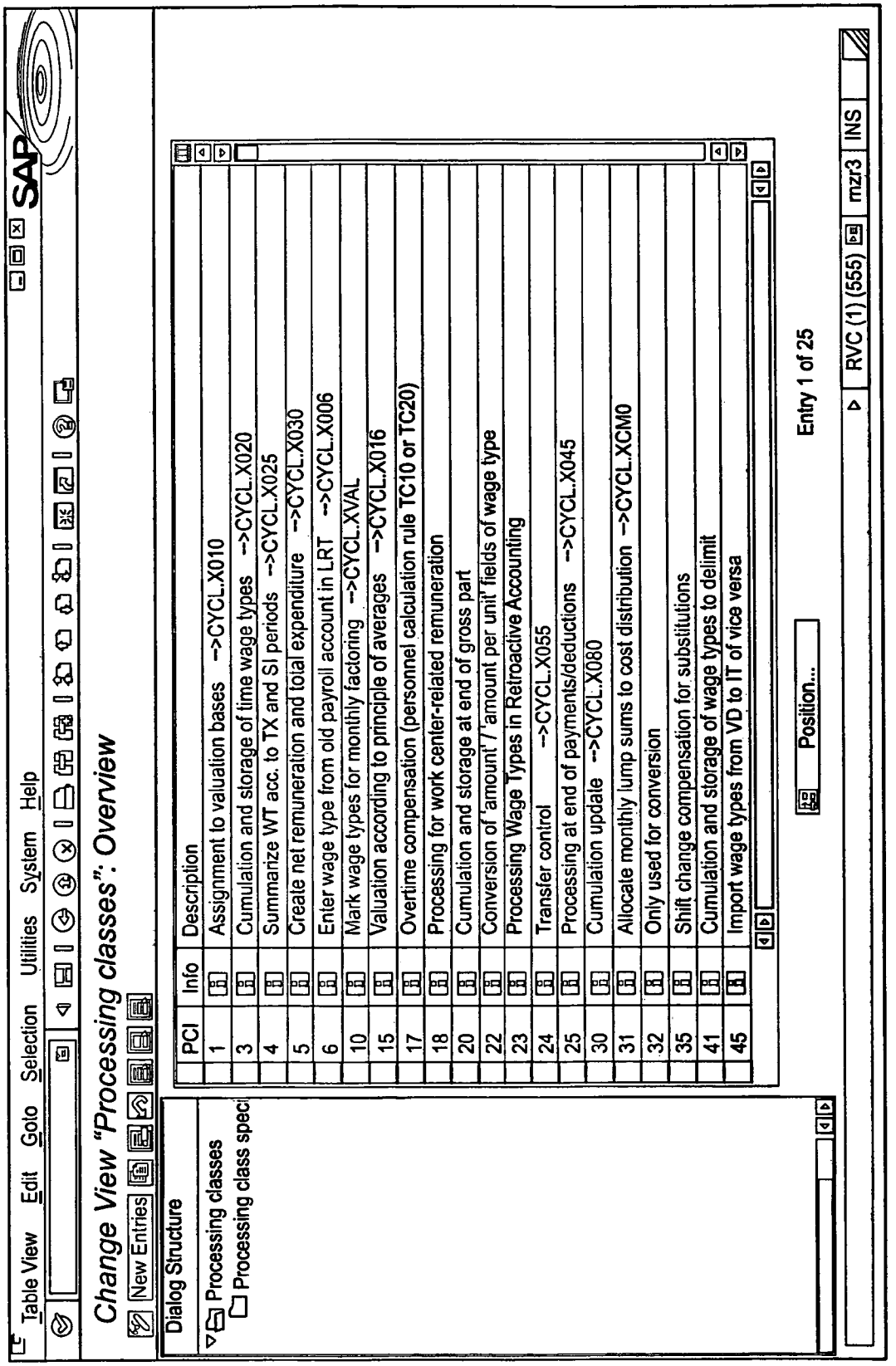


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Processing classes": Overview
New Entries
Dialog Structure
Processing classes
Processing class speci
PCl Info Description
1 Assignment to valuation bases ->CYCL.X010
3 Cumulation and storage of time wage types ->CYCL.X020
4 Summarize WT acc. to TX and SI periods ->CYCL.X025
5 Create net remuneration and total expenditure ->CYCL.X030
6 Enter wage type from old payroll account in LRT ->CYCL.X006
10 Mark wage types for monthly factoring ->CYCL.XVAL
15 Valuation according to principle of averages ->CYCL.X016
17 Overtime compensation (personnel calculation rule TC10 or TC20)
18 Processing for work center-related remuneration
20 Cumulation and storage at end of gross part
22 Conversion of 'amount' / 'amount per unit' fields of wage type
23 Processing Wage Types in Retroactive Accounting
24 Transfer control ->CYCL.X055
25 Processing at end of payments/deductions ->CYCL.X045
30 Cumulation update ->CYCL.X080
31 Allocate monthly lump sums to cost distribution ->CYCL.XCM0
32 Only used for conversion
35 Shift change compensation for substitutions
41 Cumulation and storage of wage types to delimit
45 Import wage types from VD to IT of vice versa
Position...
Entry 1 of 25
RVC (1) (555) mzr3 INS

FIG. 37

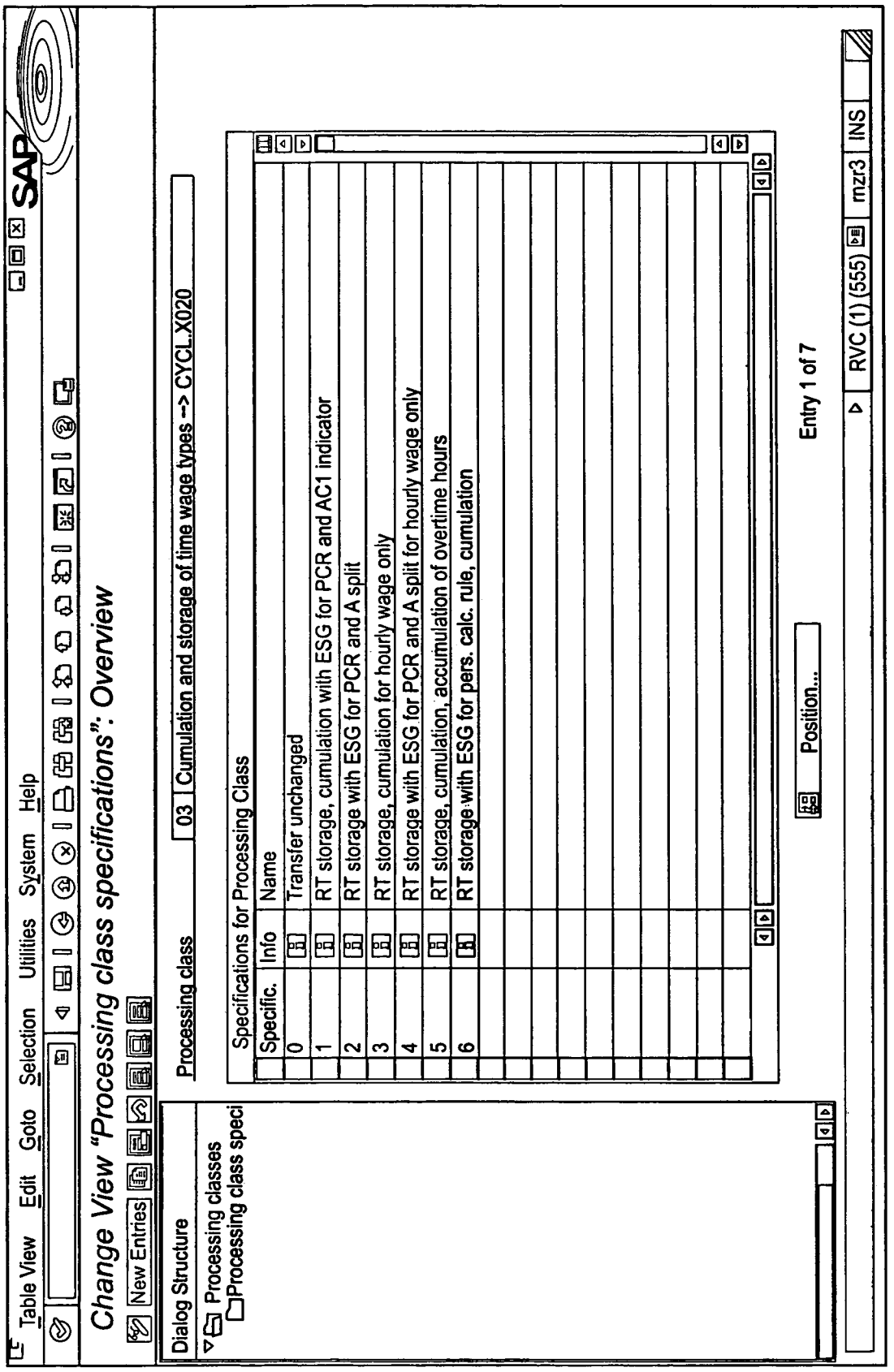


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Processing class specifications": Overview
New Entries
Dialog Structure
Processing classes
Processing class speci
Processing class
03 Cumulation and storage of time wage types --> CYCL_X020
Specifications for Processing Class
Specific.
Info
Name
0 Transfer unchanged
1 RT storage, cumulation with ESG for PCR and AC1 indicator
2 RT storage with ESG for PCR and A split
3 RT storage, cumulation for hourly wage only
4 RT storage with ESG for PCR and A split for hourly wage only
5 RT storage, cumulation, accumulation of overtime hours
6 RT storage with ESG for pers. calc. rule, cumulation
Position...
Entry 1 of 7
RVC (1) (555)
mzr3
INS

FIG. 38

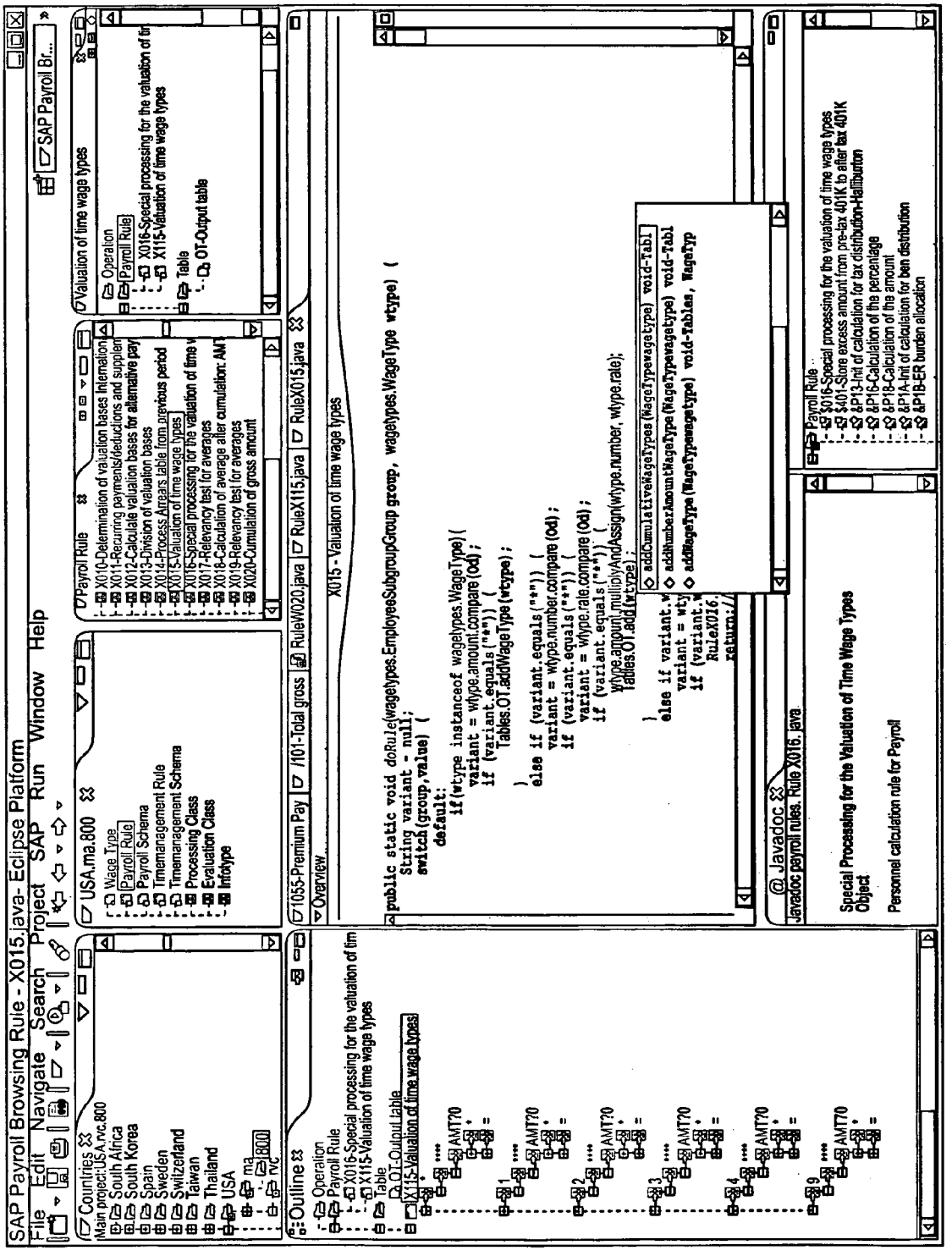

SAP Payroll Browsing Rule - X015.java - Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
SAP Payroll Br...
Countries
Main project:USA,rvc,800
South Africa
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
USA
ma
rvc
800
USA.ma.800
Wage Type
Payroll Rule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Rule
X010-Determination of valuation bases Internation
X011-Recurring payments/deductions and supplen
X012-Calculate valuation bases for alternative pay
X013-Division of valuation bases
X014-Process Arrears table from previous period
X015-Valuation of time wage types
X016-Special processing for the valuation of time w
X017-Relevancy test for averages
X018-Calculation of average after cumulation: AM
X019-Relevancy test for averages
X020-Cumulation of gross amount
Valuation of time wage types
Operation
Payroll Rule
X016-Special processing for the valuation of tir
X115-Valuation of time wage types
Table
OT-Output table
Outline
Operation
Payroll Rule
X016-Special processing for the valuation of tim
X115-Valuation of time wage types
Table
OT-Output table
X115-Valuation of time wage types
AMT70
1055-Premium Pay
/101-Total gross
RuleW020.java
RuleX115.java
RuleX015.java
Overview
X015 - Valuation of time wage types
public static void doRule(wagetypes.EmployeesSubgroupGroup group, wagetypes.WageType wtype) {
String variant = null;
switch (group,value) {
default:
if(wtype instanceof wagetypes.WageType){
variant = wtype.amount.compare (0d);
if (variant.equals("*")) {
Tables.OT.addWageType (wtype);
}
else if (variant.equals("*")) {
variant = wtype.number.compare (0d);
if (variant.equals("*")) {
variant = wtype.rate.compare (0d);
if (variant.equals("*")) {
wtype.amount.multiplyAndAssign(wtype.number, wtype.rate);
Tables.OT.add (wtype);
}
else if (variant.w
variant = wty
if (variant.w
RuleX016.
return;
addCumulativeWageTypes(WageTypewagetype) void-Tabl
addNumberAmountWageType(WageTypewagetype) void-Tabl
addWageType(WageTypewagetype) void-Tables, WageTyp
Javadoc
Javadoc payroll rules, Rule X016.java
Special Processing for the Valuation of Time Wage Types
Object
Personnel calculation rule for Payroll
Payroll Rule
$016-Special processing for the valuation of time wage types
$401-Store excess amount from pre-tax 401K to after tax 401K
&P13-Init of calculation for tax distribution-Halfbuton
&P15-Calculation of the percentage
&P18-Calculation of the amount
&P1A-Init of calculation for ben distribution
&P1B-ER burden allocation

FIG. 39

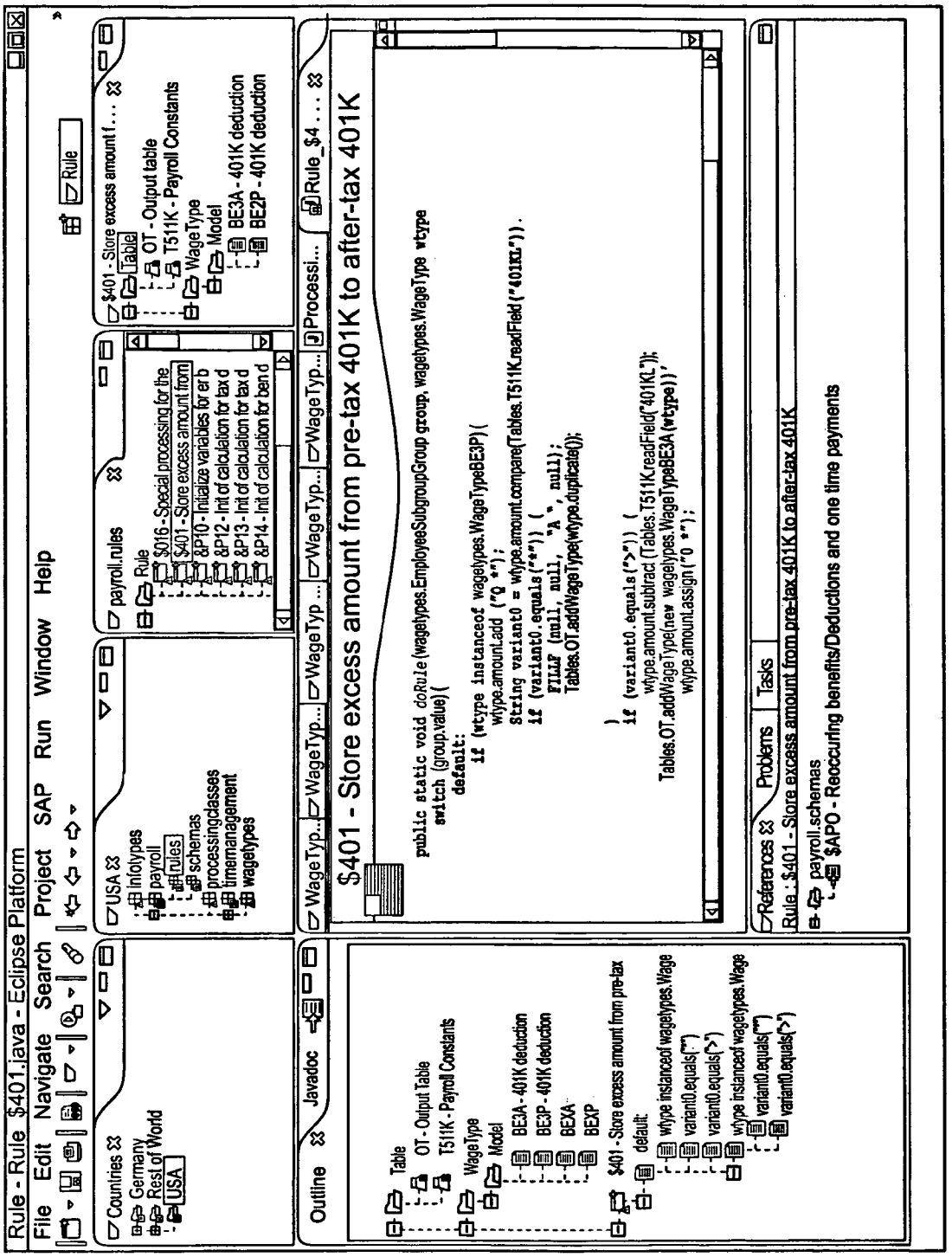

Rule - Rule $401.java - Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
Countries
Germany
Rest of World
USA
USA
InfoTypes
payroll
rules
schemas
processingclasses
timemanagement
wagetypes
payroll.rules
Rule
$016 - Special processing for the
$401 - Store excess amount from
&P10 - Initialize variables for er b
&P12 - Init of calculation for tax d
&P13 - Init of calculation for tax d
&P14 - Init of calculation for ben d
$401 - Store excess amount f...
Table
OT - Output table
T511K - Payroll Constants
WageType
Model
BE3A - 401K deduction
BE2P - 401K deduction
Rule
Outline
Javadoc
WageTyp...
WageTyp...
WageTyp ...
WageTyp...
WageTyp...
Processi...
Rule_$4 ...
$401 - Store excess amount from pre-tax 401K to after-tax 401K
public static void doRule(wagetypes.EmployeeSubgroupGroup group, wagetypes.WageType wtype
switch (group.value){
default:
if (wtype instanceof wagetypes.WageTypeBE2P) {
wtype.amountLadd ("0 **");
String variant0 = wtype.amount.compare(Tables.T511K.readField("401KL")).
if (variant0.equals("*") {
FIELF (null, null, "A ", null);
Tables.OT.addWageType(wtype.duplicate());
}
if (variant0.equals(">")) {
wtype.amount.subtract (Tables.T511K.readField("401KL"));
Tables.OT.addWageType(new wagetypes.WageTypeBE3A (wtype)) '
wtype.amount.assign ("0 **");
References
Problems
Tasks
Rule : $401 - Store excess amount from pre-tax 401K to after-tax 401K
payroll.schemas
$APO - Reoccuring benefits/Deductions and one time payments
Table
OT - Output Table
T511K - Payroll Constants
WageType
Model
BE3A - 401K deduction
BE2P - 401K deduction
BE3A
BE2P
$401 - Store excess amount from pre-tax
default:
wtype instanceof wagetypes.Wage
variant0.equals("")
variant0.equals(">")
wtype instanceof wagetypes.Wage
variant0.equals("")
variant0.equals(">")

FIG. 40

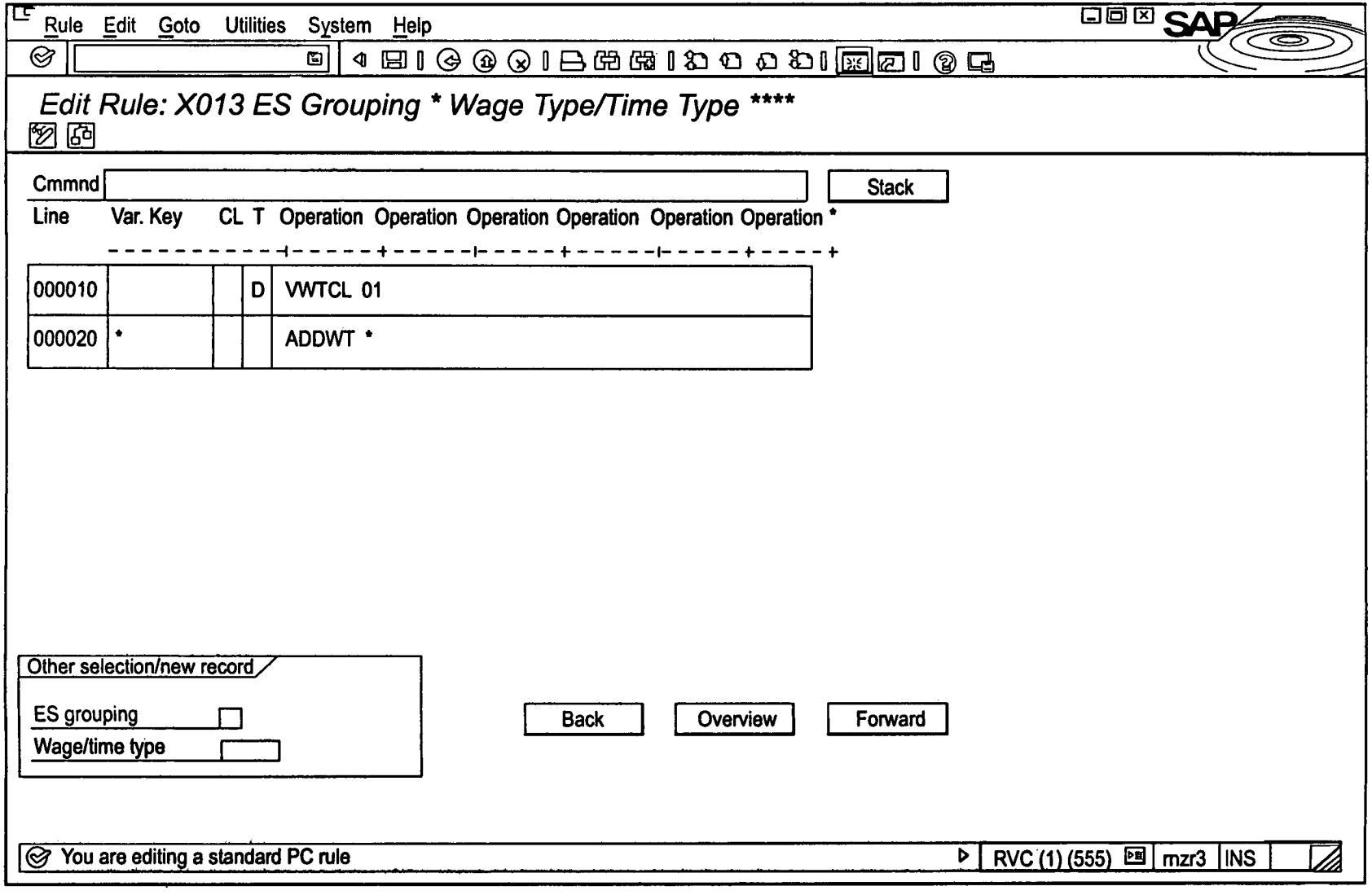

Rule Edit Goto Utilities System Help
SAP
Edit Rule: X013 ES Grouping * Wage Type/Time Type ****
Cmmnd
Stack
Line Var. Key CL T Operation Operation Operation Operation Operation Operation *
000010 D VWTCL 01
000020 * ADDWT *
Other selection/new record
ES grouping
Wage/time type
Back
Overview
Forward
You are editing a standard PC rule
RVC (1) (555)
mzr3
INS

FIG. 41

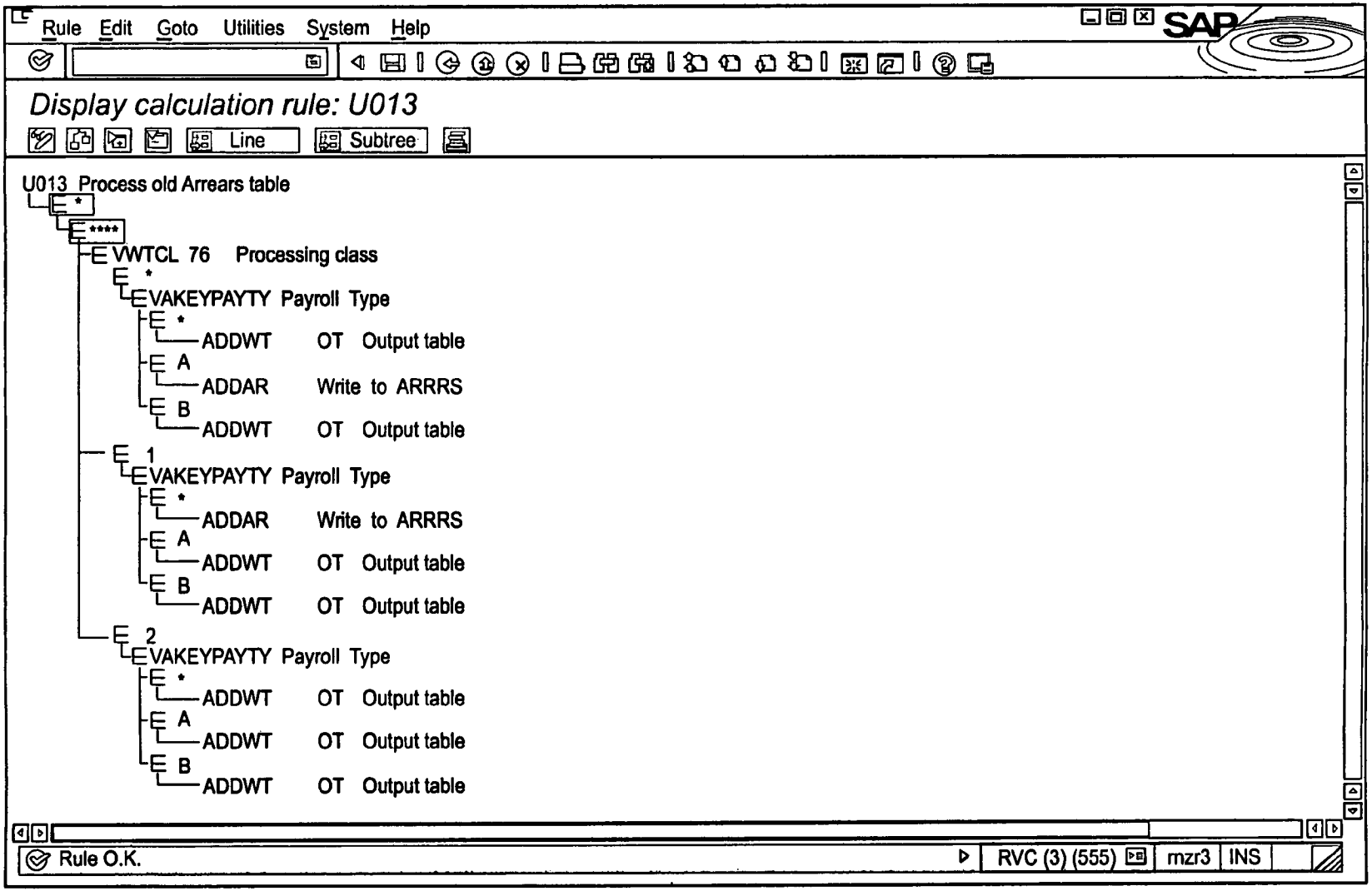

Rule Edit Goto Utilities System Help
SAP
Display calculation rule: U013
Line
Subtree
U013 Process old Arrears table
*
****
VWTCL 76 Processing class
*
VAKEYPAYTY Payroll Type
*
ADDWT OT Output table
A
ADDAR Write to ARRRS
B
ADDWT OT Output table
1
VAKEYPAYTY Payroll Type
*
ADDAR Write to ARRRS
A
ADDWT OT Output table
B
ADDWT OT Output table
2
VAKEYPAYTY Payroll Type
*
ADDWT OT Output table
A
ADDWT OT Output table
B
ADDWT OT Output table
Rule O.K.
RVC (3) (555)
mzr3
INS

FIG. 42

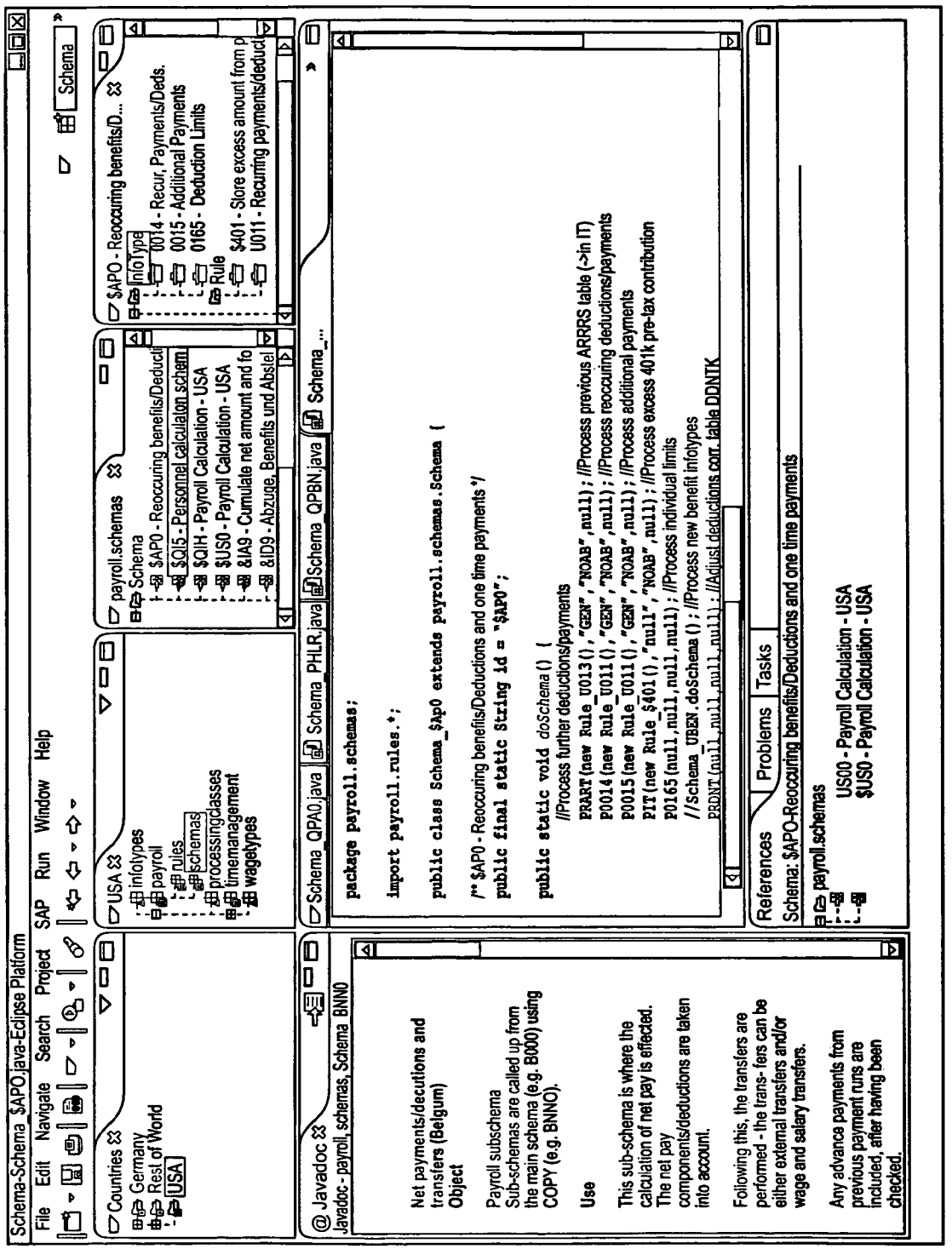
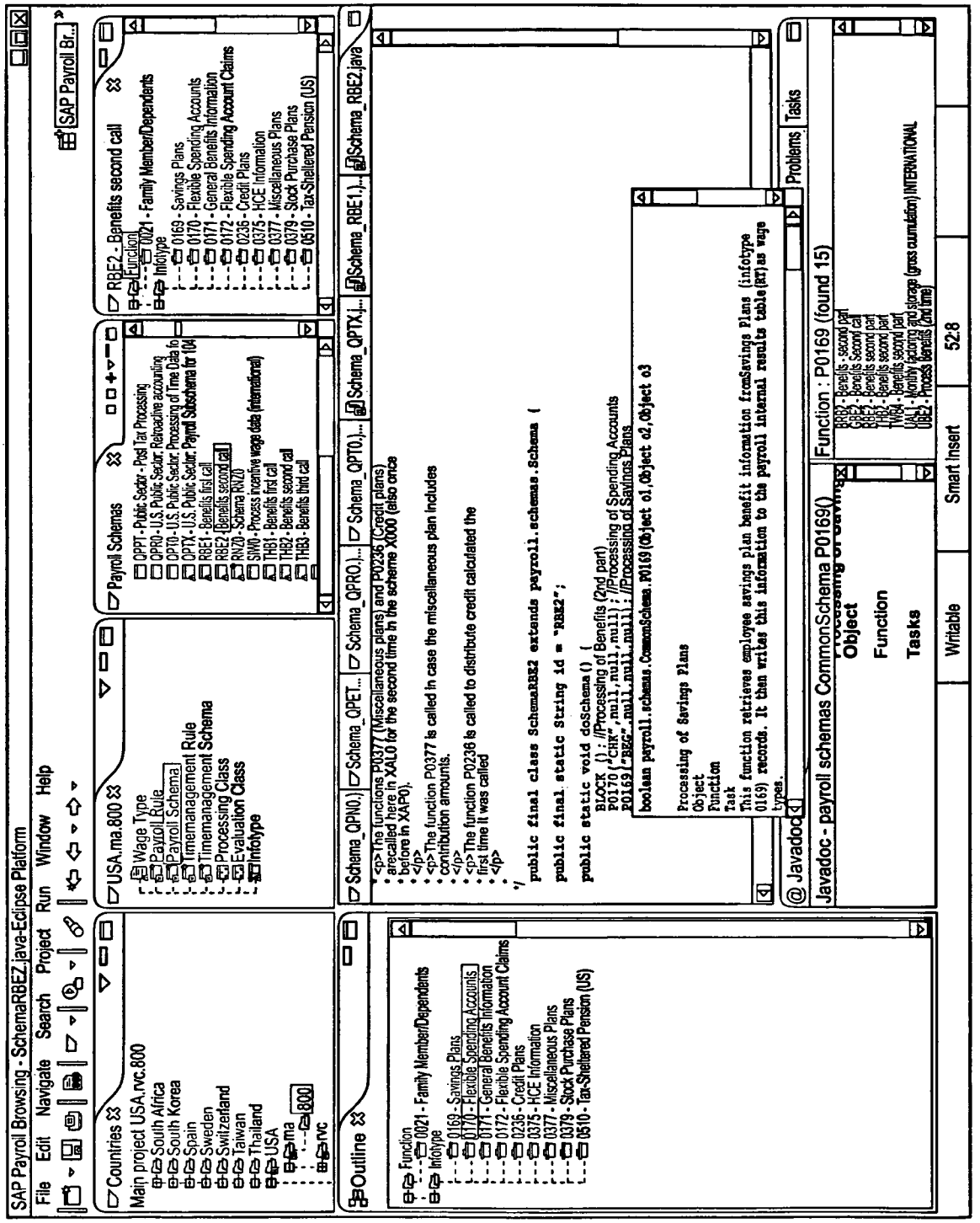

Schema-Schema_$APO/java-Eclipse Platform
File Edit Navigate Search Project SAP Run Window Help
Schema
Countries
Germany
Rest of World
USA
USA
infotypes
payroll
rules
schemas
processingclasses
timemanagement
wagetypes
payroll.schemas
Schema
$APO - Reoccuring benefits/Deducti
$QI5 - Personnel calculation schem
$QIH - Payroll Calculation - USA
$US0 - Payroll Calculation - USA
&IA9 - Cumulate net amount and fo
&ID9 - Abzuge, Benefits und Abstel
$APO - Reoccuring benefits/D...
InfoType
0014 - Recur, Payments/Deds.
0015 - Additional Payments
0165 - Deduction Limits
Rule
$401 - Store excess amount from p
U011 - Recurring payments/deduct
Javadoc
Javadoc - payroll.schemas.Schema BNNO
Net payments/decutions and transfers (Belgum)
Object
Payroll subschema
Sub-schemas are called up from the main schema (e.g. B000) using COPY (e.g. BNNO).
Use
This sub-schema is where the calculation of net pay is effected. The net pay components/deductions are taken into account.
Following this, the transfers are performed - the trans- fers can be either external transfers and/or wage and salary transfers.
Any advance payments from previous payment runs are included, after having been checked.
Schema_QPA0.java
Schema_PHLR.java
Schema_QPBN.java
Schema_...
package payroll.schemas;
import payroll.rules.*;
public class Schema_$ap0 extends payroll.schemas.Schema {
/** $AP0 - Reoccuring benefits/Deductions and one time payments */
public final static String id = "$AP0";
public static void doSchema() {
//Process further deductions/payments
PRART(new Rule_U013(),"GEN","NOAB",null); //Process previous ARRRS table (->in IT)
P0014(new Rule_U011(),"GEN","NOAB",null); //Process reoccuring deductions/payments
P0015(new Rule_U011(),"GEN","NOAB",null); //Process additional payments
PIT(new Rule_§401(),"null","NOAB",null); //Process excess 401k pre-tax contribution
P0165(null,null,null,null); //Process individual limits
//Schema_UBEN.doSchema(); //Process new benefit infotypes
PRONT(null,null,null,null); //Adjust deductions com. table DDNTK
References
Problems
Tasks
Schema: $APO-Reoccuring benefits/Deductions and one time payments
payroll.schemas
US00 - Payroll Calculation - USA
$US0 - Payroll Calculation - USA

FIG. 43


SAP Payroll Browsing - SchemaRBE2.java-Eclipse Platform
File Edit Navigate Search Project Run Window Help
SAP Payroll Br...
Countries
Main project USA.rvc.800
South Africa
South Korea
Spain
Sweden
Switzerland
Taiwan
Thailand
USA
ma
800
rvc
USA.ma.800
Wage Type
PayrollRule
Payroll Schema
Timemanagement Rule
Timemanagement Schema
Processing Class
Evaluation Class
Infotype
Payroll Schemas
QPPT - Public Sector - Post Tax Processing
QPR0 - U.S. Public Sector. Retroactive accounting
QPT0 - U.S. Public Sector. Processing of Time Data fo
QPTX - U.S. Public Sector. Payroll Subschema for 104
RBE1 - Benefits first call
RBE2 - Benefits second call
RWZ0 - Schema RWZ0
SIW0 - Process incentive wage data (international)
THB1 - Benefits first call
THB2 - Benefits second call
THB3 - Benefits third call
RBE2 - Benefits second call
Function
0021 - Family Member/Dependents
Infotype
0169 - Savings Plans
0170 - Flexible Spending Accounts
0171 - General Benefits Information
0172 - Flexible Spending Account Claims
0236 - Credit Plans
0375 - HCE Information
0377 - Miscellaneous Plans
0379 - Stock Purchase Plans
0510 - Tax-Sheltered Pension (US)
Outline
Function
0021 - Family Member/Dependents
Infotype
0169 - Savings Plans
0170 - Flexible Spending Accounts
0171 - General Benefits Information
0172 - Flexible Spending Account Claims
0236 - Credit Plans
0375 - HCE Information
0377 - Miscellaneous Plans
0379 - Stock Purchase Plans
0510 - Tax-Sheltered Pension (US)
Schema_QPN0.j...
Schema_QPET...
Schema_QPR0.j...
Schema_QPT0.j...
Schema_QPTX.j...
Schema_RBE1.j...
Schema_RBE2.java
The functions P0377 (Miscellaneous plans) and P0236 (Credit plans)
are called here in XAL0 for the second time in the scheme X000 (also once
before in XAP0).

The function P0377 is called in case the miscellaneous plan includes
contribution amounts.

The function P0236 is called to distribute credit calculated the
first time it was called

*/
public final class SchemaRBE2 extends payroll.schemas.Schema {
public final static String id = "RBE2";
public static void doSchema() {
BLOCK (); //Processing of Benefits (2nd part)
P0170 ("CHK", null, null, null); //Processing of Spending Accounts
P0169 ("RBE", null, null, null); //Processing of Savings Plans
boolean payroll.schemas.CommonSchema.P0169(Object o1,Object o2,Object o3
Processing of Savings Plans
Object
Function
Task
This function retrieves employee savings plan benefit information fromSavings Plans (infotype
0169) records. It then writes this information to the payroll internal results table(RT) as wage
types.
Javadoc
Problems
Tasks
Javadoc - payroll schemas CommonSchema P0169()
Function : P0169 (found 15)
Object
Function
Tasks
RBE2 - Benefits second part
RBE2 - Benefits second call
RBE2 - Benefits second part
THB3 - Benefits second part
THB4 - Benefits second part
UAL1 - Monthly updating and storage (gross cumulation) INTERNATIONAL
UBE2 - Process benefits (2nd time)
Writable
Smart Insert
52:8

FIG. 44

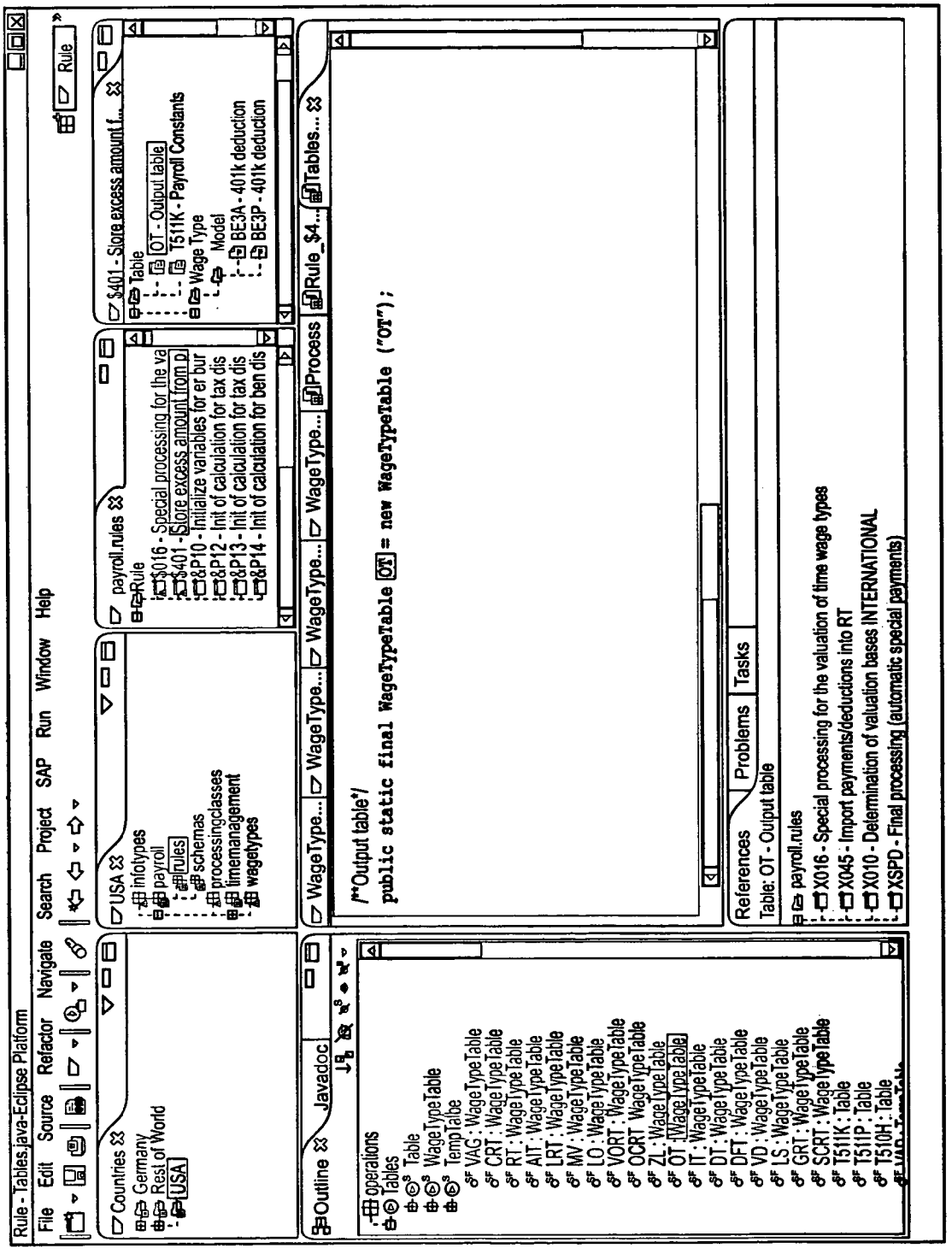

Rule - Tables.java-Eclipse Platform
File Edit Source Refactor Navigate Search Project SAP Run Window Help
Rule
Countries
Germany
Rest of World
USA
USA
infotypes
payroll
rules
schemas
processingclasses
timemanagement
wagetypes
payroll.rules
Rule
$016 - Special processing for the va
$401 - Store excess amount from p
&P10 - Initialize variables for er bur
&P12 - Init of calculation for tax dis
&P13 - Init of calculation for tax dis
&P14 - Init of calculation for ben dis
$401 - Store excess amount f...
Table
OT - Output table
T511K - Payroll Constants
Wage Type
Model
BE3A - 401k deduction
BE3P - 401k deduction
WageType...
WageType...
WageType...
Process
Rule_$4...
Tables...
/**Output table*/
public static final WageTypeTable OT = new WageTypeTable ("OT");
Outline
Javadoc
operations
Tables
Table
WageTypeTable
TempTable
VAG : WageTypeTable
CRT : WageTypeTable
RT : WageTypeTable
AIT : WageTypeTable
LRT : WageTypeTable
MV : WageTypeTable
LO : WageTypeTable
VORT : WageTypeTable
OCRT WageTypeTable
ZL : WageTypeTable
OT : WageTypeTable
IT : WageTypeTable
DT : WageTypeTable
DFT : WageTypeTable
VD : WageTypeTable
LS : WageTypeTable
GRT : WageTypeTable
SCRT : WageTypeTable
T511K : Table
T511P : Table
T510H : Table
References
Problems
Tasks
Table: OT - Output table
payroll.rules
X016 - Special processing for the valuation of time wage types
X045 - Import payments/deductions into RT
X010 - Determination of valuation bases INTERNATIONAL
XSPD - Final processing (automatic special payments)

FIG. 45

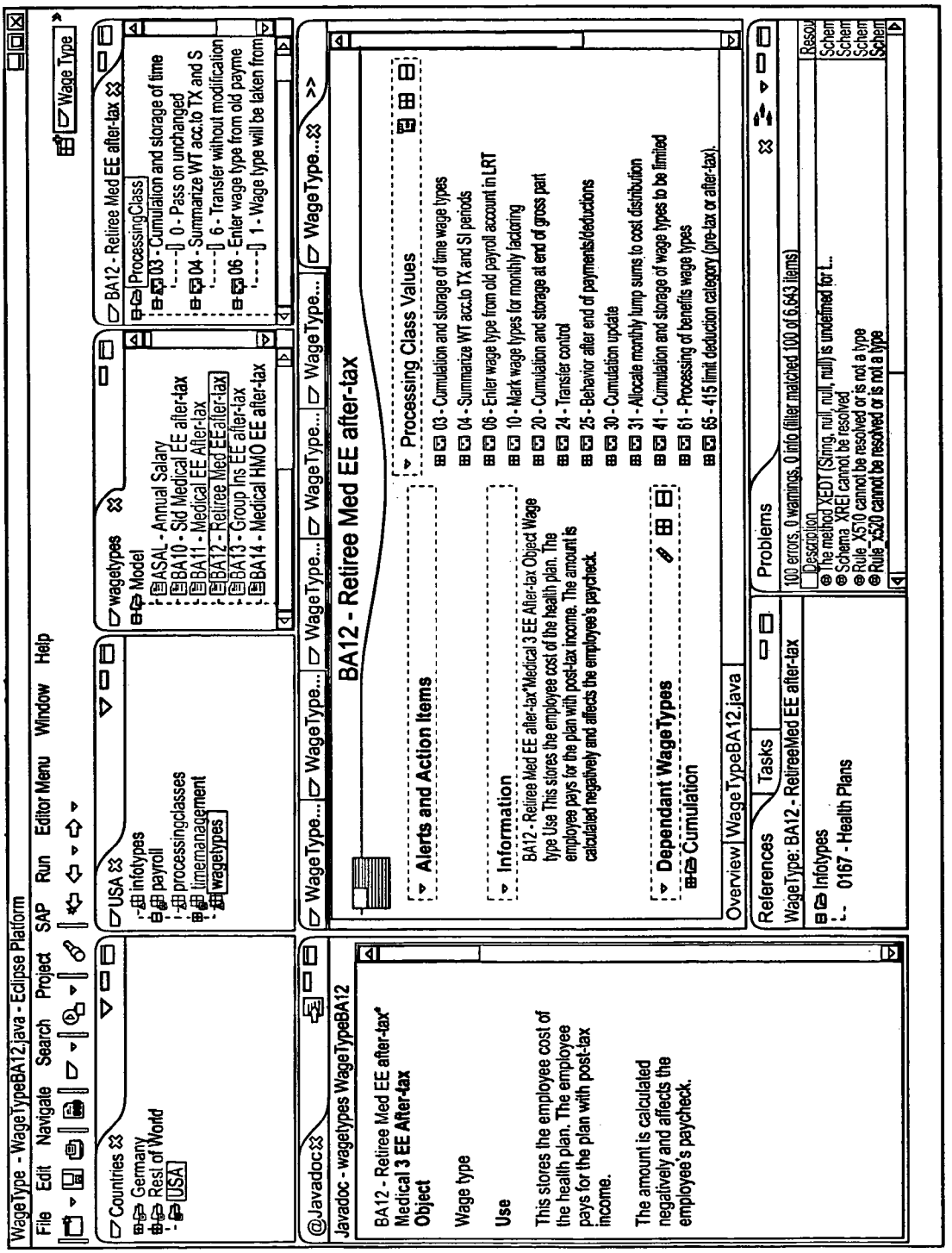

WageType - WageTypeBA12.java - Eclipse Platform
File Edit Navigate Search Project SAP Run Editor Menu Window Help
Wage Type
Countries
Germany
Rest of World
USA
USA
infotypes
payroll
processingclasses
timemanagement
wagetypes
wagetypes
Model
ASAL - Annual Salary
BA10 - Std Medical EE after-tax
BA11 - Medical EE After-tax
BA12 - Retiree Med EE after-tax
BA13 - Group Ins EE after-tax
BA14 - Medical HMO EE after-tax
BA12 - Retiree Med EE after-tax
ProcessingClass
03 - Cumulation and storage of time
0 - Pass on unchanged
04 - Summarize WT acc.to TX and S
6 - Transfer without modification
06 - Enter wage type from old payme
1 - Wage type will be taken from
@Javadoc
Javadoc - wagetypes WageTypeBA12
BA12 - Retiree Med EE after-tax* Medical 3 EE After-tax
Object
Wage type
Use
This stores the employee cost of the health plan. The employee pays for the plan with post-tax income.
The amount is calculated negatively and affects the employee's paycheck.
WageType...
WageType...
WageType...
WageType...
BA12 - Retiree Med EE after-tax
Alerts and Action Items
Information
BA12 - Retiree Med EE after-tax* Medical 3 EE After-tax Object Wage type Use This stores the employee cost of the health plan. The employee pays for the plan with post-tax income. The amount is calculated negatively and affects the employee's paycheck.
Dependant WageTypes
Cumulation
Processing Class Values
03 - Cumulation and storage of time wage types
04 - Summarize WT acc.to TX and SI periods
06 - Enter wage type from old payroll account in LRT
10 - Mark wage types for monthly factoring
20 - Cumulation and storage at end of gross part
24 - Transfer control
25 - Behavior after end of payments/deductions
30 - Cumulation update
31 - Allocate monthly lump sums to cost distribution
41 - Cumulation and storage of wage types to be limited
61 - Processing of benefits wage types
65 - 415 limit deduction category (pre-tax or after-tax).
Overview
WageTypeBA12.java
References
Tasks
WageType: BA12 - RetireeMed EE after-tax
Infotypes
0167 - Health Plans
Problems
100 errors, 0 warnings, 0 info (filter matched 100 of 6,643 items)
Description
The method XEDT (String, null, null, null) is undefined for t...
Schema_XREI cannot be resolved
Rule_X510 cannot be resolved or is not a type
Rule_x520 cannot be resolved or is not a type
Resou
Schem
Schem
Schem
Schem

FIG. 46

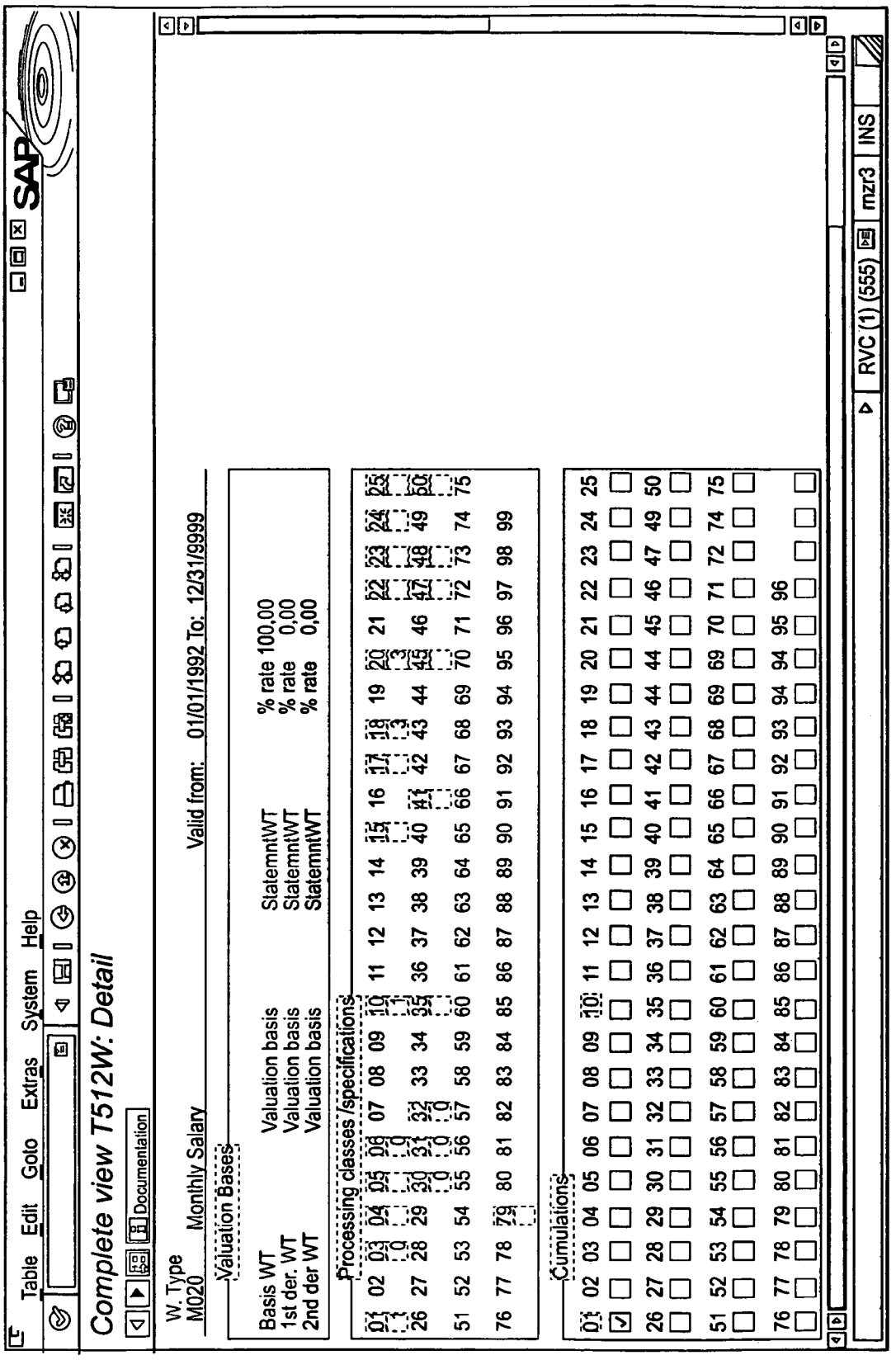


Table Edit Goto Extras System Help
SAP
Complete view T512W: Detail
Documentation
W Type
M020 Monthly Salary
Valid from: 01/01/1992 To: 12/31/9999
Valuation Bases
Basis WT Valuation basis StatemntWT % rate 100,00
1st der. WT Valuation basis StatemntWT % rate 0,00
2nd der WT Valuation basis StatemntWT % rate 0,00
Processing classes /specifications
Cumulations
RVC (1) (555)
mzr3
INS

FIG. 47

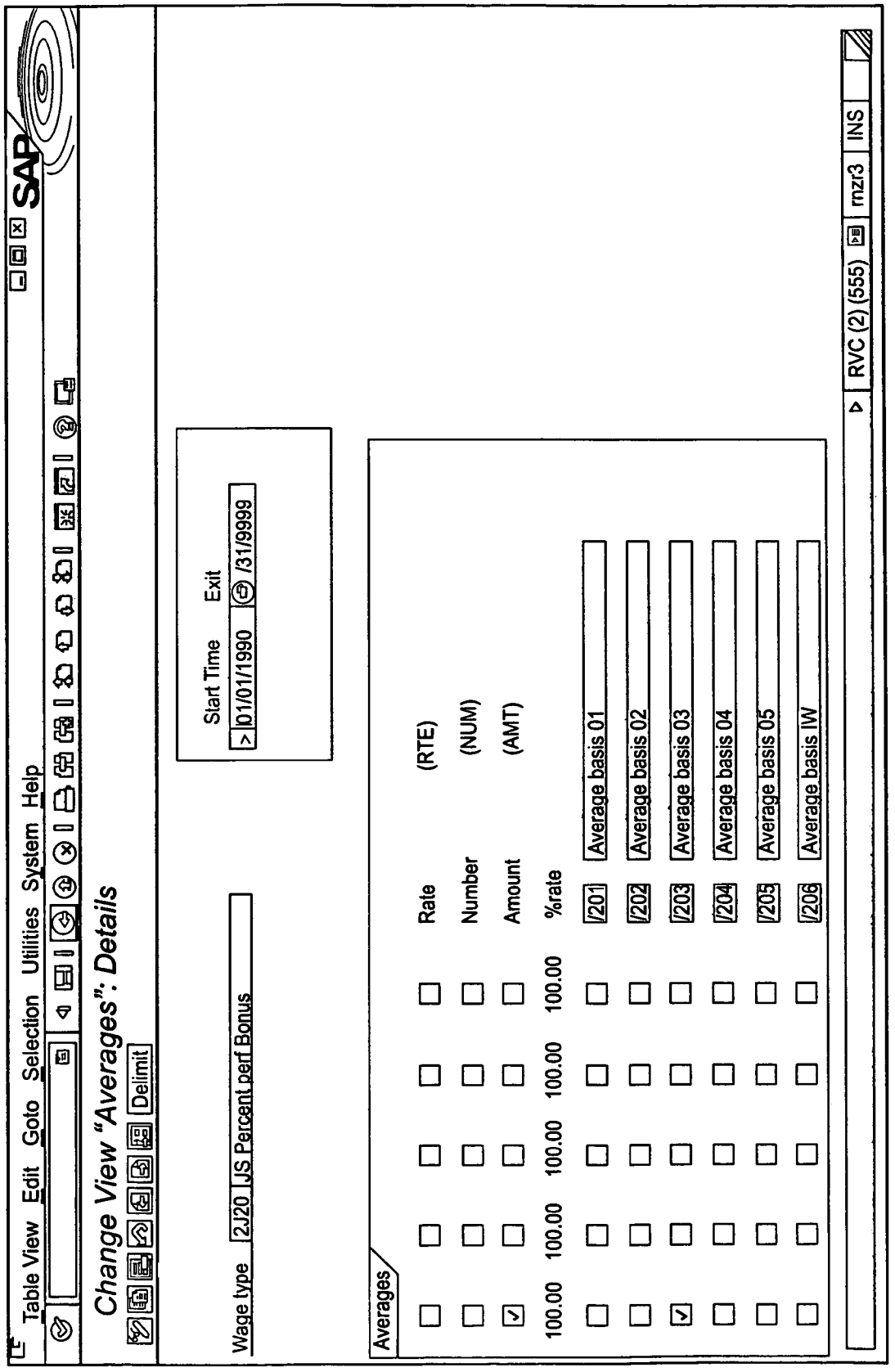


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Averages": Details
Delimit
Wage type 2J20 JS Percent perf Bonus
Start Time Exit
> 01/01/1990 /31/9999
Averages
Rate (RTE)
Number (NUM)
Amount (AMT)
%rate
100.00 100.00 100.00 100.00 100.00
/201 Average basis 01
/202 Average basis 02
/203 Average basis 03
/204 Average basis 04
/205 Average basis 05
/206 Average basis IW
RVC (2) (555)
rnzr3
INS

FIG. 48

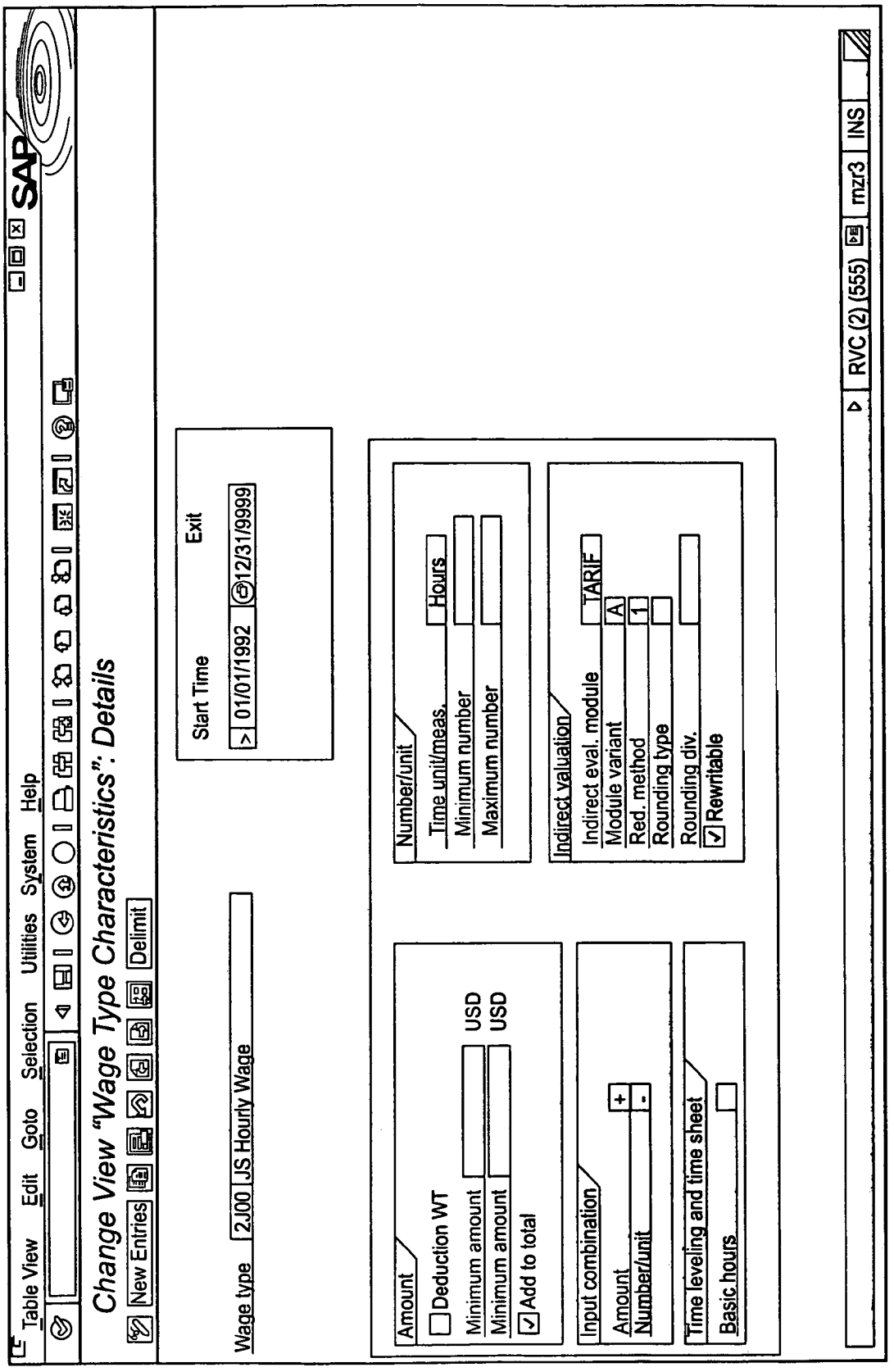


Table View Edit Goto Selection Utilities System Help
SAP
Change View "Wage Type Characteristics": Details
New Entries
Delimit
Wage type 2J00 JS Hourly Wage
Start Time
Exit
01/01/1992
12/31/9999
Amount
Deduction WT
Minimum amount
USD
Minimum amount
USD
Add to total
Number/unit
Time unit/meas.
Hours
Minimum number
Maximum number
Indirect valuation
Indirect eval. module
TARIF
Module variant
A
Red. method
1
Rounding type
Rounding div.
Rewritable
Input combination
Amount
Number/unit
Time leveling and time sheet
Basic hours
RVC (2) (555)
mzr3
INS

FIG. 49

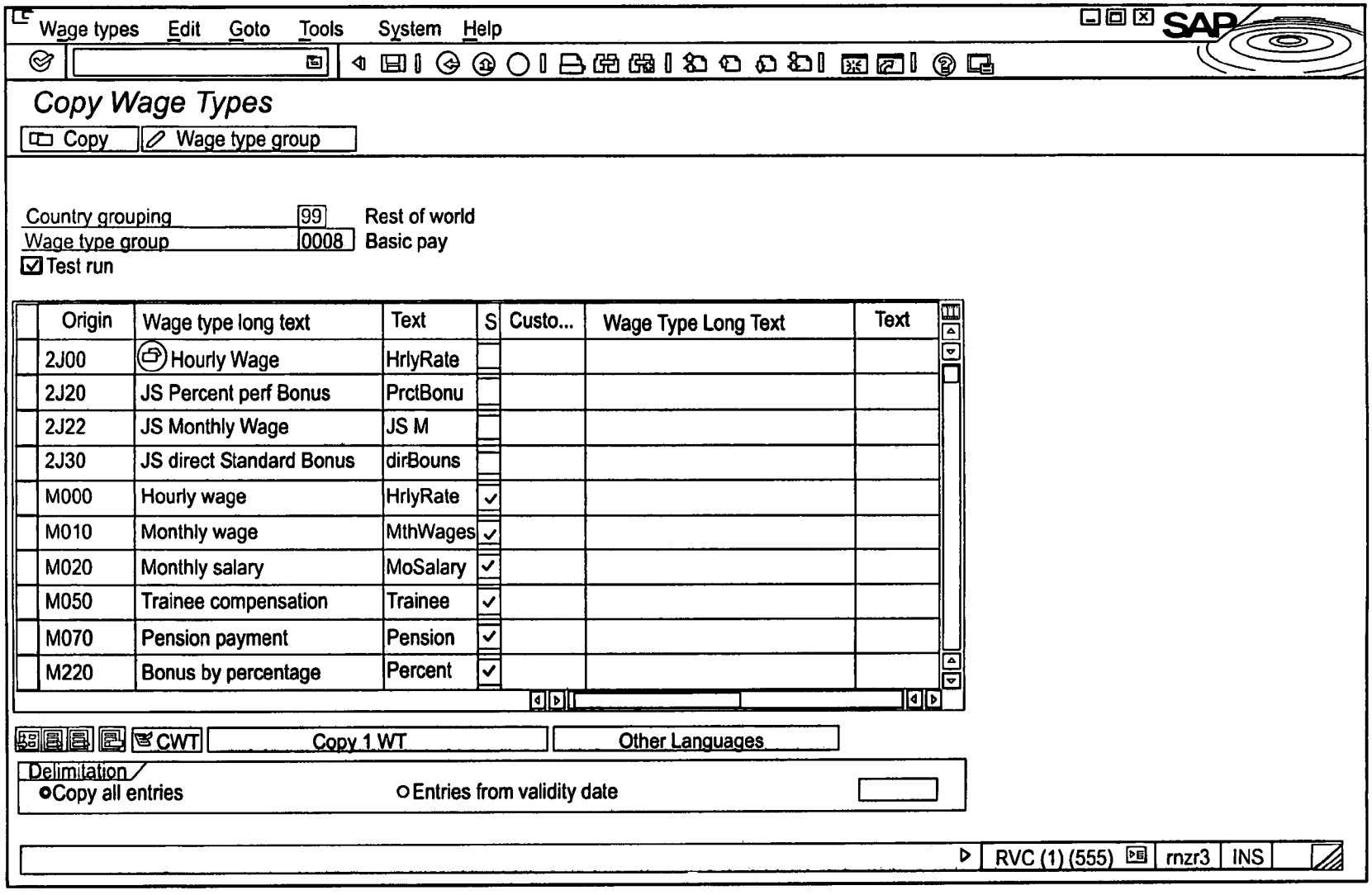

Wage types Edit Goto Tools System Help
SAP
Copy Wage Types
Copy
Wage type group
Country grouping 99 Rest of world
Wage type group 0008 Basic pay
Test run
Origin
Wage type long text
Text
S
Custo...
Wage Type Long Text
Text
2J00 Hourly Wage HrlyRate
2J20 JS Percent perf Bonus PrctBonu
2J22 JS Monthly Wage JS M
2J30 JS direct Standard Bonus dirBouns
M000 Hourly wage HrlyRate
M010 Monthly wage MthWages
M020 Monthly salary MoSalary
M050 Trainee compensation Trainee
M070 Pension payment Pension
M220 Bonus by percentage Percent
CWT
Copy 1 WT
Other Languages
Delimitation
Copy all entries
Entries from validity date
RVC (1) (555)
rnzr3
INS

FIG. 50

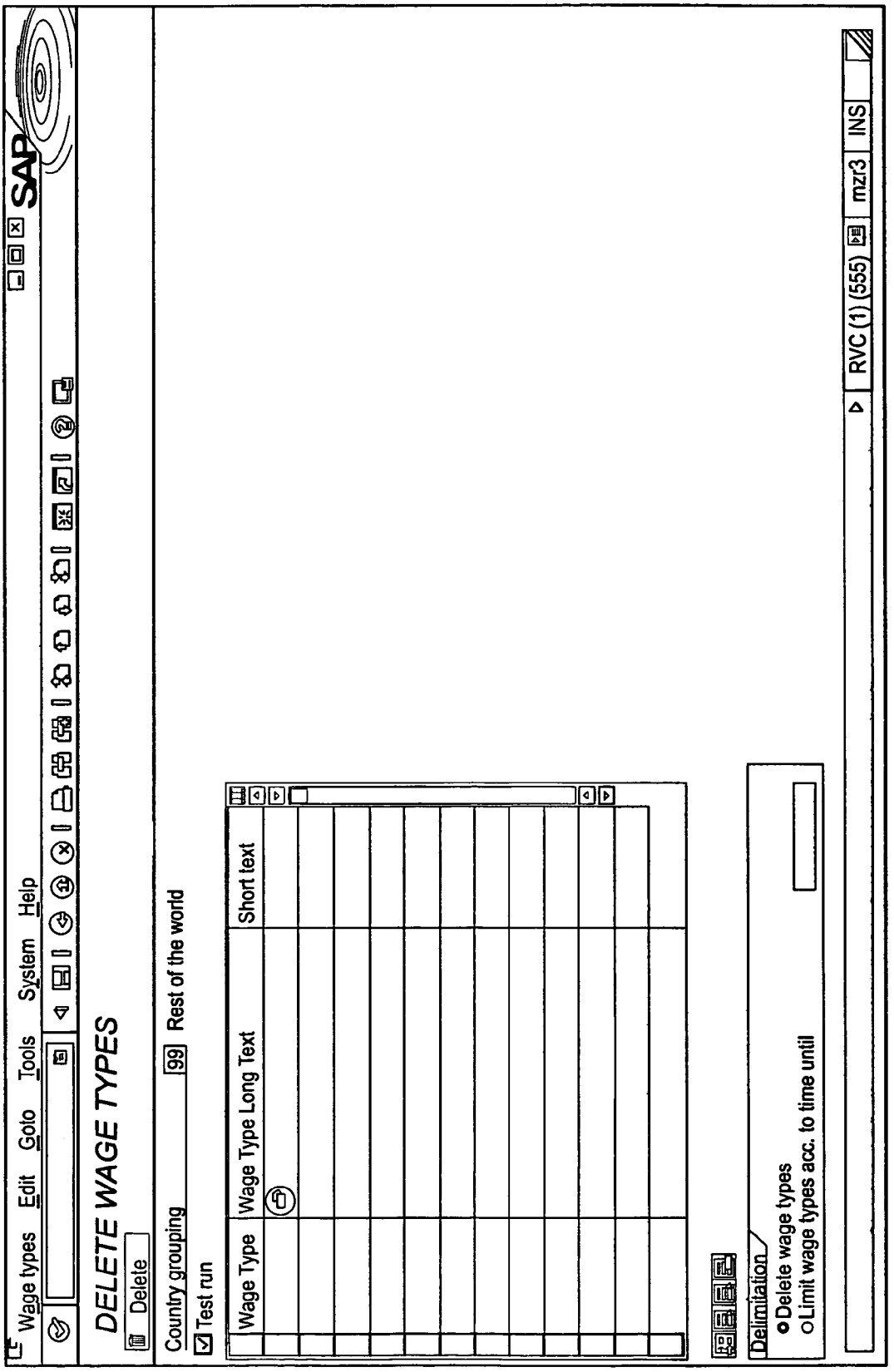

Wage types Edit Goto Tools System Help
SAP
DELETE WAGE TYPES
Delete
Country grouping
99
Rest of the world
Test run
Wage Type
Wage Type Long Text
Short text
Delimitation
Delete wage types
Limit wage types acc. to time until
RVC (1) (555)
mzr3
INS

FIG. 51

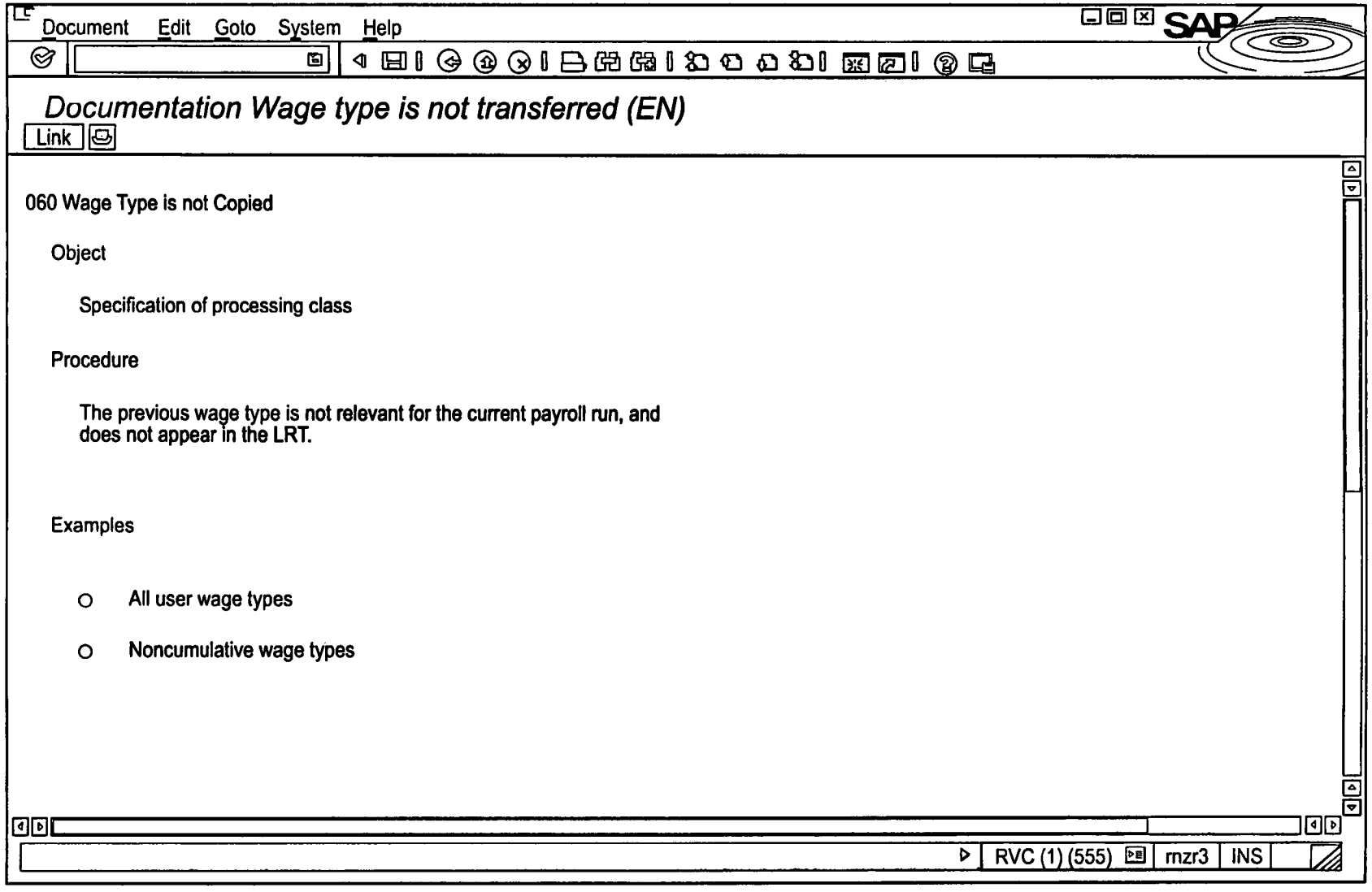

Document Edit Goto System Help
SAP
Documentation Wage type is not transferred (EN)
Link
060 Wage Type is not Copied
Object
Specification of processing class
Procedure
The previous wage type is not relevant for the current payroll run, and does not appear in the LRT.
Examples
All user wage types
Noncumulative wage types
RVC (1) (555)
rnzr3
INS

FIG. 52

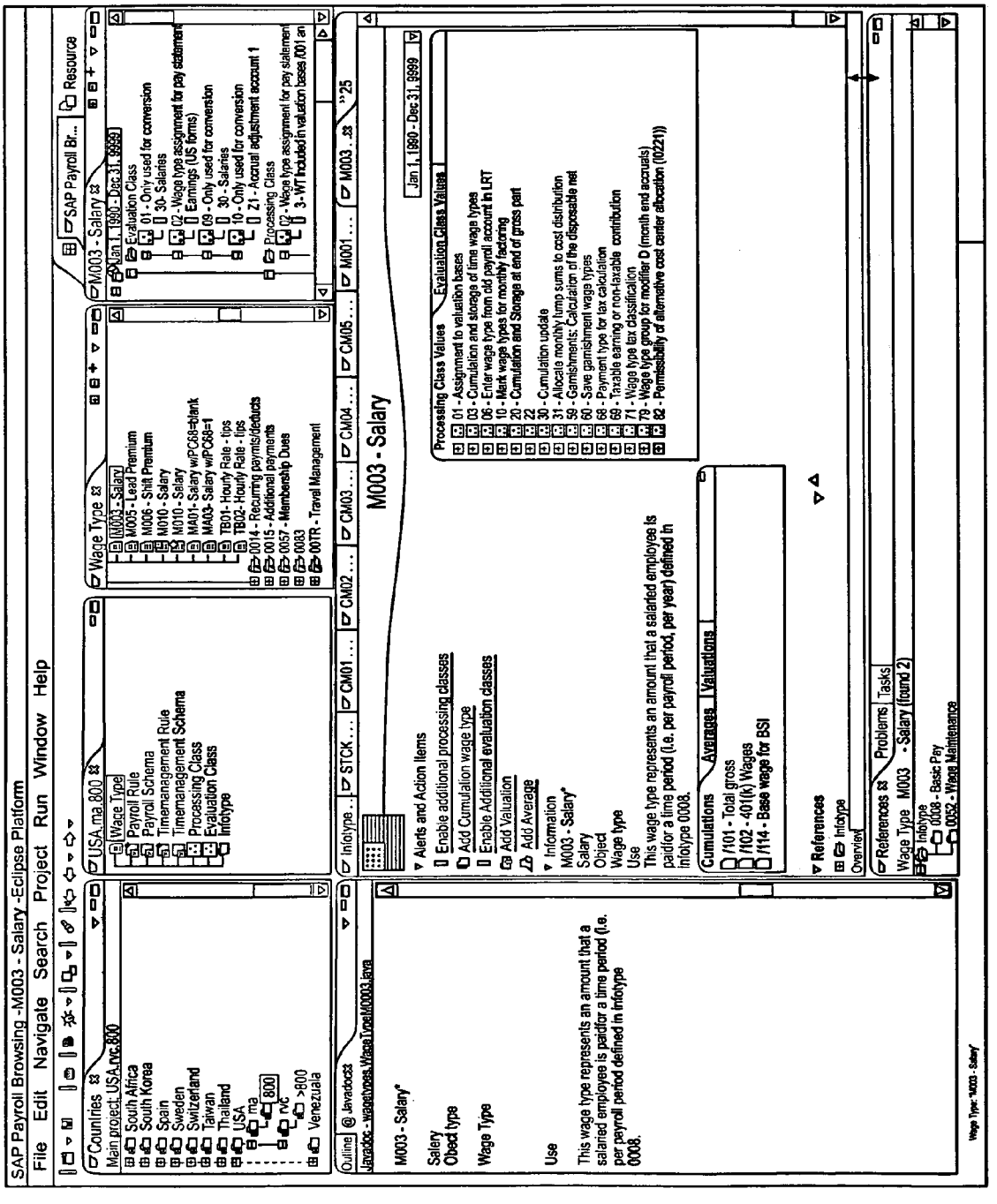

SAP Payroll Browsing -M003 - Salary -Eclipse Platform
File Edit Navigate Search Project Run Window Help
M003 - Salary
Alerts and Action Items
Enable additional processing classes
Add Cumulation wage type
Enable Additional evaluation classes
Add Valuation
Add Average
Information
M003 - Salary*
Salary
Object
Wage type
Use
This wage type represents an amount that a salaried employee is paid/or a time period (i.e. per payroll period, per year) defined in infotype 0008.
Processing Class Values
Evaluation Class Values
Cumulations
Averages
Valuations
References
Problems
Tasks

FIG. 53

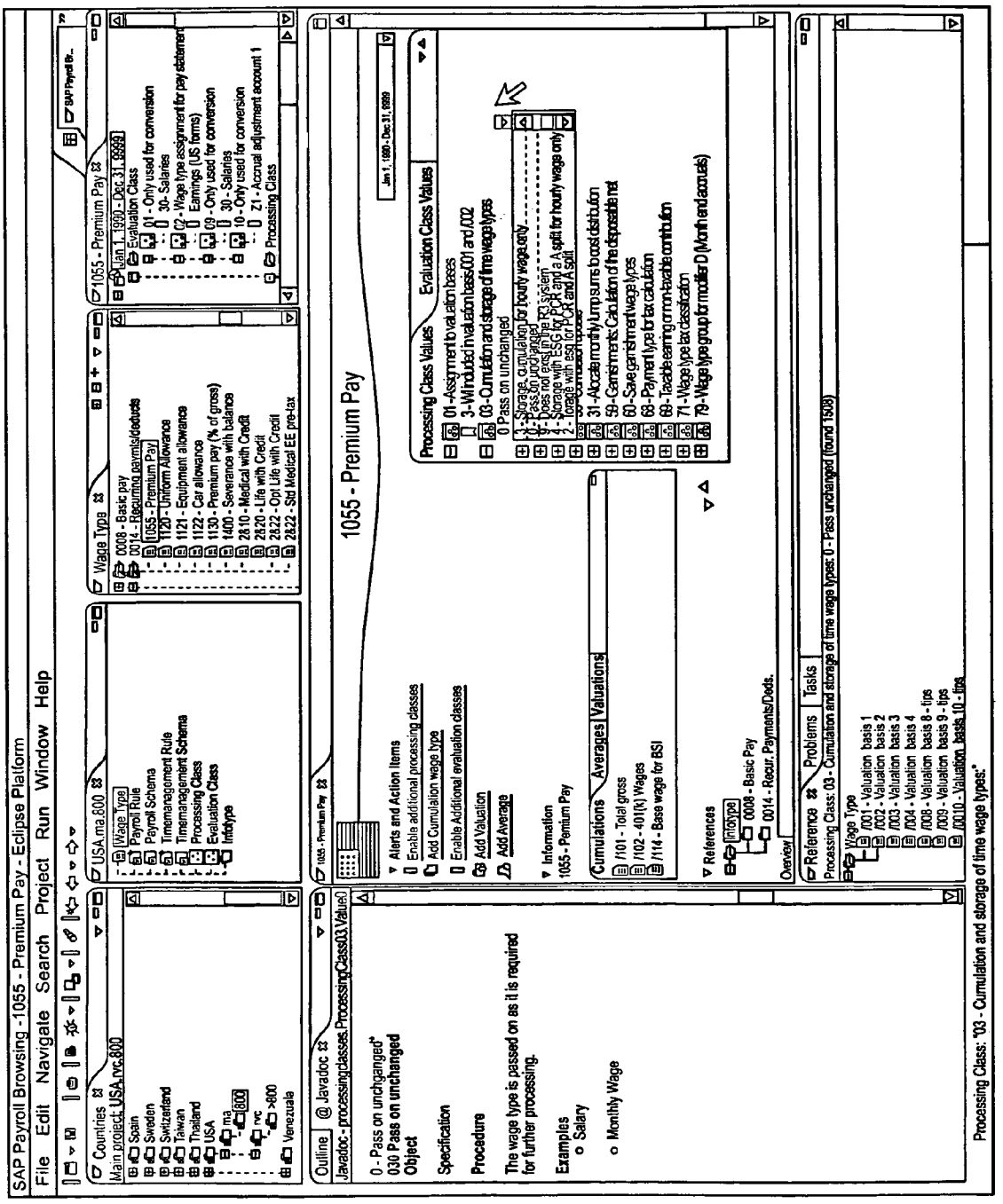

SAP Payroll Browsing - 1055 - Premium Pay - Eclipse Platform
File Edit Navigate Search Project Run Window Help
1055 - Premium Pay
Processing Class: '03 - Cumulation and storage of time wage types:'

FIG. 54

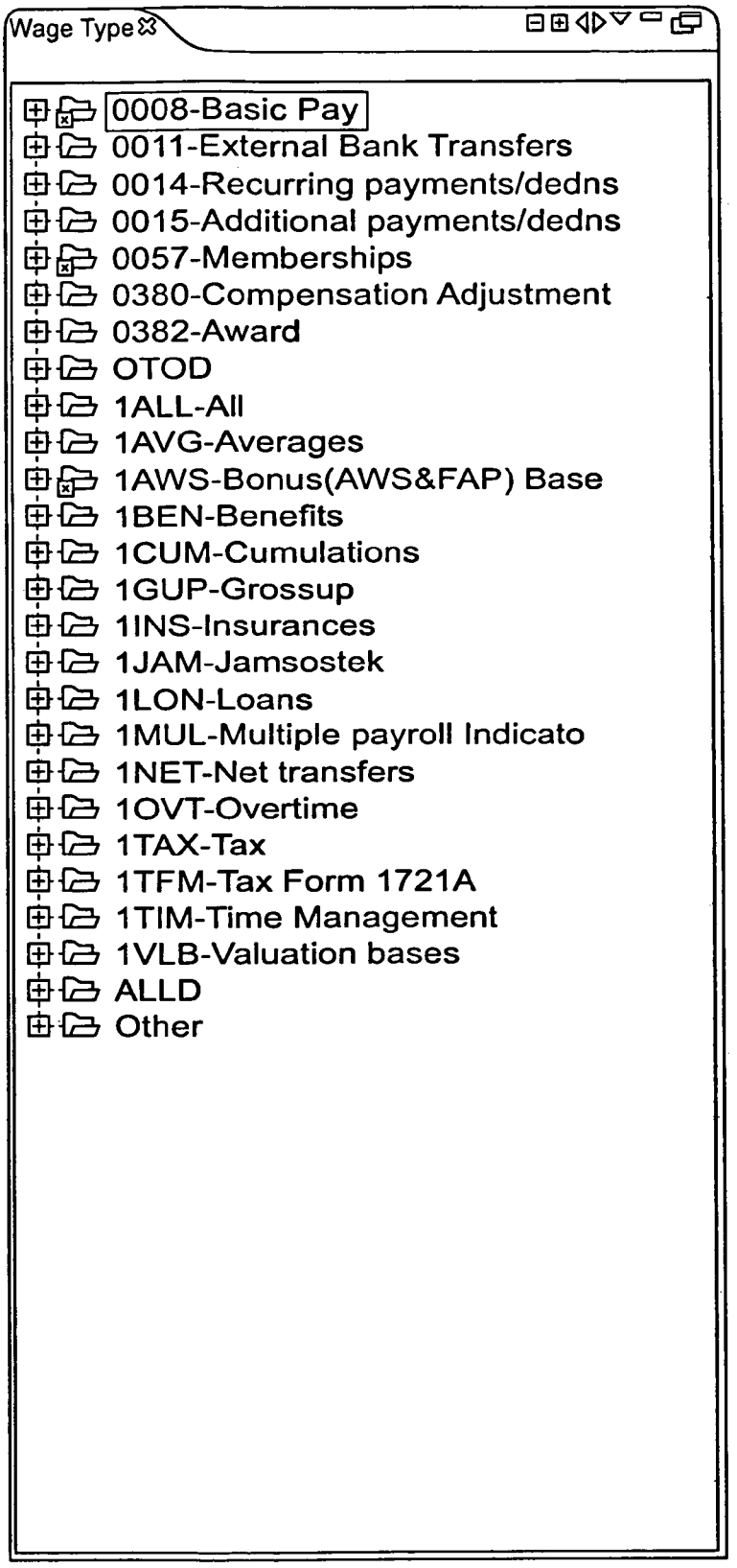

Wage Type
0008-Basic Pay
0011-External Bank Transfers
0014-Recurring payments/dedns
0015-Additional payments/dedns
0057-Memberships
0380-Compensation Adjustment
0382-Award
OTOD
1ALL-All
1AVG-Averages
1AWS-Bonus(AWS&FAP) Base
1BEN-Benefits
1CUM-Cumulations
1GUP-Grossup
1INS-Insurances
1JAM-Jamsostek
1LON-Loans
1MUL-Multiple payroll Indicato
1NET-Net transfers
1OVT-Overtime
1TAX-Tax
1TFM-Tax Form 1721A
1TIM-Time Management
1VLB-Valuation bases
ALLD
Other

FIG. 55

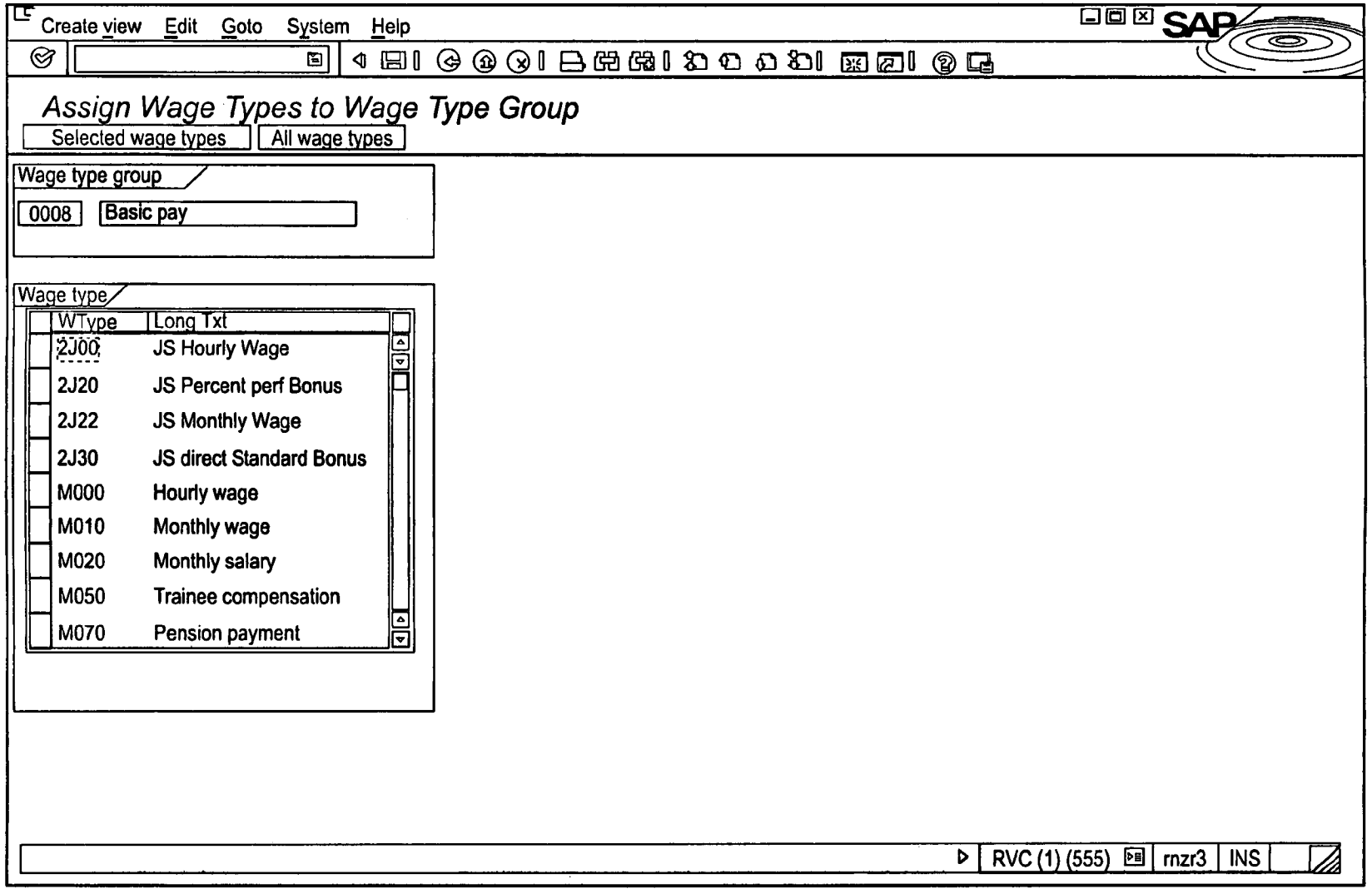

Create view Edit Goto System Help
SAP
Assign Wage Types to Wage Type Group
Selected wage types
All wage types
Wage type group
0008
Basic pay
Wage type
WType
Long Txt
2J00
JS Hourly Wage
2J20
JS Percent perf Bonus
2J22
JS Monthly Wage
2J30
JS direct Standard Bonus
M000
Hourly wage
M010
Monthly wage
M020
Monthly salary
M050
Trainee compensation
M070
Pension payment
RVC (1) (555)
rnzr3
INS

FIG. 56

METHODS OF COMPARING AND MERGING BUSINESS PROCESS CONFIGURATIONS

RELATED APPLICATION

This application is a continuation-in-part of U.S. patent application Ser. No. 11/111,794, entitled "Systems and Methods for Modeling and Manipulating a Table-Driven Business Application in an Object-Oriented Environment," filed Apr. 22, 2005, which is incorporated herein by reference.

BACKGROUND

1. Relevant Field

The present inventions generally relate to business application software. More particularly, the inventions relate to systems, methods and computer readable media for object-oriented programming (OOP) modeling of business applications and the use of an OOP model to create, improve, and maintain a business application.

2. Background Information

One drawback with legacy business application systems stems from how business data and rules are often contained in physically separate table data structures, with little user-friendly representation of the organization or relationships of those structures. Such table-driven systems are difficult to work with. For example, a business object may be defined across ten or more different database tables that are not easily accessed, displayed, or understood together.

Furthermore, as shown in the example of FIG. **3**A, a table-driven business application typically stores its configuration data **305** and its rules **310** in multiple, physically separate tables. Considered statically, there is no obvious or easily determined relationship or connection **315** between configuration data **305** and rules **310**. At runtime, however, configuration data **305** and rules **310** interrelate **320** because the configuration data **305** and the rules **310** affect each other. For example, some configuration data **305**, may affect the execution of rules **310**. The physical separation between related data and between data and rules, the lack of static indications of runtime interactions between data and rules, and other factors make the data and processes of a business application difficult to understand, difficult to create, difficult to debug, and difficult to modify without causing unexpected effects.

In view of the foregoing, it is desirable to take business application structures and translate them into user-friendly objects that can be more easily understood, manipulated, debugged, designed, redesigned, analyzed, and/or modified.

SUMMARY

Consistent with embodiments of the present inventions, systems, methods and computer readable media are disclosed for modeling of business applications and the use of an OOP model to create, improve, and maintain a business application.

Embodiments and implements consistent with the invention provide methods and systems of determining a current version of a business application. The methods and systems perform operations comprising: receiving a first version of an object-oriented model representing the business application, wherein the first version includes a plurality of objects; receiving a second version of the object-oriented model representing the business application, wherein the second version includes a plurality of objects; receiving a third version of the object-oriented model representing the business application, wherein the third version includes a plurality of objects; comparing the objects of the first version, the second version, and the third version of the object-oriented model to identify differences between the first version, the second version, and the third version of the object-oriented model; determining, for each identified difference, which version of the object-oriented model is to be associated with the first version of the object-oriented model; and assigning the first version of the object-oriented model as a current version of the object-oriented model.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only, and should not be considered restrictive of the scope of the invention, as described and claimed. Further, features and/or variations may be provided in addition to those set forth herein. For example, embodiments of the invention may be directed to various combinations and sub-combinations of the features described in the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate various embodiments and aspects of the present invention. In the drawings:

FIGS. **1** to **3**B illustrate exemplary business structures of a business application;

FIG. **4**A illustrates a flowchart illustrating an overview of an exemplary process **400** for creating an off-line model of a business application, consistent with the present invention;

FIG. **4**B is an exemplary illustration **440** of the various processing tools consistent with the invention;

FIGS. **5**, **6**A, and **6**B illustrate exemplary processes consistent with the present invention;

FIG. **7**A illustrate an exemplary processes consistent with the present invention;

FIG. **7**B illustrates an exemplary OOP structure consistent with the invention;

FIG. **7**C illustrates a block diagram of an exemplary architecture consistent with an embodiment of the invention;

FIG. **8** illustrates an exemplary processes consistent with the present invention;

FIGS. **9** to **15** illustrate exemplary processing tool operations consistent with the present invention;

FIG. **16** illustrates an exemplary process consistent with the invention; and

FIGS. **17** to **56** illustrate exemplary user interface displays consistent with the present invention.

DETAILED DESCRIPTION

The following detailed description refers to the accompanying drawings. Wherever possible, the same reference numbers are used in the drawings and the following description to refer to the same or similar parts. While several exemplary embodiments and features of the invention are described herein, modifications, adaptations and other implementations are possible, without departing from the spirit and scope of the invention. For example, substitutions, additions or modifications may be made to the components illustrated in the drawings, and the exemplary methods described herein may be modified by substituting, reordering, or adding steps to the disclosed methods. Accordingly, the following detailed description does not limit the invention. Instead, the proper scope of the invention is defined by the appended claims.

Overview

Systems and methods consistent with the invention generally relate to business application software. Exemplary business applications include those offered by SAP, built around the SAP R/3 system. These business applications may provide the capability to manage financial asset and cost accounting, as well as production operations and materials, personnel, plants, and archived documents. The R/3 system runs on a number of platforms including Windows 2000 and uses the client/server model. The R/3 system is a comprehensive set of integrated business applications. R/3 provides the ability to store, retrieve, analyze, and process corporate data used for financial analysis, production operation, human resource management, and other business processes. At one level, an R/3 system (or other legacy business application system) may be thought of as a collection of legacy business structures or business objects held primarily in databases (as tables) that are manipulated by business processes or workflows.

For example, as shown in FIG. **1**A, an exemplary business structure includes configuration data **100**, which may be one or more business data objects containing data used by a business process. Configuration data **100** may include several data fields called attributes, such as "attribute **1**" **105** through "attribute n" **120**. For example, configuration data **100** may include data such as an employee's name, an hourly wage rate, and indicators for directing processing of data for the employee. Exemplary business structures also include business rules, such as rule **125**, which are typically made up of multiple operations, such as "operation **1**" **130** through "operation n" **140**. Rule **125** may reflect a series of operations that when executed will, for example, cease deducting social security payments after an employee's cumulative salary reaches a specified amount.

In systems such as R/3, configuration data and rules control the business application software. As shown in FIG. **2**A, in an exemplary payroll processing business application, the business structures may include a wagetype **200**, which may be a business data object containing configuration data used to configure portions of a paycheck. For payroll processing purposes, each employee may have associated with him or her one or more wagetype structures containing data about the employee that is used to perform the payroll functions. For example, an employee may have a wagetype for hourly rate, a wagetype for union fees, a wagetype for deductions, etc. Each wagetype may include one or more attributes **205** that act as configuration elements, shown as attributes A-H. The attributes themselves may be of different types. For example, as shown, attributes A-C may be processing class attributes, which control the processing path used by the business application when performing processing (rules) for the employee. Attributes D-F may be cumulation attributes, which identify wage types that relate to one another and are processed together in some instances. Attributes G and H may be evaluation class attributes, which control processing paths like processing class attributes, but for different rules. A wage type **200** may include other types of attributes **205**, as well.

As shown in FIG. **2**B, as one of its functions, an exemplary payroll processing application **215** takes employee data **210** as input and produces a paycheck **220** as output. At runtime, each employee record is processed based on the employee's data **210** to produce the employee's paycheck **220**.

FIG. **2**C illustrates the exemplary process of FIG. **2**B in greater detail. As shown in this example, employee data **210** is contained in the employee's wagetype(s) **200** or in one or more similarly organized data structures. These data structures contain the data used by, and to control, process **215** that calculates and produces a paycheck **220**.

More specifically, employee data **210** from the employee wage type **200** may be used by a workflow or schema that implements payroll process **215**. In the example shown, the first function of payroll process **215** may be the "HRS×RATE" function **225**, which may multiply the employee's hours worked by the employee's hourly wage rate. The data specifying the hours and hourly rate may be derived from the employee's wage type **200**. As shown in this example, information regarding how to perform the "HRS×RATE" function **225** is specified by a rule(s) **226** (rule "A" in FIG. **2**C) associated with function **225**. The functioning of the rule(s) **226**, and thus the functioning of the "HRS×RATE" function **225**, may be affected by attributes **205** contained in the employee's wage type **200**. For example, as shown in FIG. **2**A, a processing class attribute **205**, such as attribute A, may have "L" selected for this employee and "R" unselected. Selecting "L" may cause rule(s) **226** associated with the "HRS×RATE" function **225** to calculate the product in a different manner than if "R" were selected for attribute A. For example, "L" may cause the hourly rate to increase by 1.5 for hours greater than 40, such that the employee gets paid time and a half for overtime beyond 40 hours, while "R" would not use an hourly rate increase, for an employee who is not eligible for increased overtime pay.

As shown in FIG. **2**C, the next function to be performed in the workflow may be either the "ADD BONUS" function **230** or the "SUBTRACT TAXES" function **235**. Typically, data from the employee's wage type **200** will indicate whether the employee is to receive a bonus and control the workflow to determine which of the two functions is performed next. As explained with respect to the "HRS×RATE" function **225**, the rules associated with the "ADD BONUS" function **230** or the "SUBTRACT TAXES" function **235** (rules "B" and "C," respectively, of FIG. **2**C) determine how those functions are performed, and attributes **205** from the employee's wage type **200** may affect the functioning of the rules.

Similarly, the "SUBTRACT DEDUCTIONS" function **240**, "SUBTRACT ALIMONY" function **245**, and "CUMULATE" function **250** are performed as the illustrated workflow progresses, and the rules associated with each function may be similarly affected by the corresponding attributes **205** in the employee's wage type **200**. As shown in FIG. **2**C, rules "D," "E," and "F" correspond to functions **240**, **245**, and **250**, respectively. After all the workflow functions are completed, the payroll process shown produces a paycheck **220** for the employee reflecting the results of the operations carried out by the various functions.

Other business applications, such as financial applications, asset accounting applications, cost accounting applications, production operations applications, etc., use business structures and workflows comparable to the payroll processing application illustrated in FIGS. **2**A-**2**C.

FIG. **3**B illustrates an exemplary business system architecture to which embodiments of the invention may be applied. As shown, a business system **330**, such as an SAP R/3 system, may include several business applications, such as business application A **340**, business application B **350**, and business application C **360**. A business application, may be, for example, a payroll application that calculates the payroll for a company, generates paychecks or payslips for each employee, etc.; a human resources application that manages employee data, adds new employee records, deactivates retired employee records, etc.; or some other type of business application.

A typical legacy business application, such as a payroll application exemplified in this example by business application A **340**, is table-driven. In other words, the data which controls execution of the application is contained in database tables. The application is designed to read and traverse the tables and execute business functions or processes according to the contents of the database tables. Multiple applications run on the same physical hardware and from data contained in the same relational database(s).

At a high level, payroll business application A **340** may be thought of as consisting of basically three major logical parts: (1) a data dictionary **342** that describes all the tables related to the application, (in this case payroll), (2) source code **344** that is the driver of the business processes within the business application, and (3) configuration data **346** that control the operation of the business processes within the business application. Each application in business system **330**, such as business application B **350** and business application C **360**, typically has its own data dictionary, source code, and configuration data. Configuration data is typically very application specific, while some data dictionary business objects and some source code may be shared among business applications.

Data dictionary **342** contains information about the business that is used by the business application, such as information about the employees used by the payroll application, for example, information regarding when a new employee joined the company. Configuration data **346** contains information controlling the operation of business application A **340** and is typically stored in database tables accessible to users who can customize the payroll application for their business needs by modifying configuration data **346**. Some embodiments consistent with the invention focus on modeling configuration data **346**, which controls the operation of business application source code **344**, such as the payroll driver.

In one embodiment, configuration data **346** may be logically modeled by several major business objects. For example, an SAP R/3 business system payroll application may be modeled as major business objects such as wagetypes, schemas, and rules. Further, major business objects may be logically modeled as containing one or more sub-objects such as processing class, evaluation class, and cumulations, for an R/3 wagetype. The configuration data in the modeled objects describes how the business application is configured for a specific user, such as a specific company.

Source code **344** represents a driver for the business processes within the business application. The application source code driver of the business application is essentially just a framework that needs the configuration data to tell it how to operate. Without the configuration data to tell it how to work, the source code cannot work meaningfully. The source code driver **344** knows how to execute any schema, which is part of configuration data **346**, and a schema is required to run an application; e.g., a payroll schema is required to run a payroll business application.

Payroll driver **344** knows how to read a schema table and do what the schema table says. A schema may be considered a work flow description. For example, in an R/3 system, the business application source code, such as payroll driver source code **344**, is basically designed to navigate or traverse the configuration database tables that stores a schema(s). Business application driver **344** reads the schema table row by row, takes the data out of the schema tables, and executes the rules associated with a specific function of the schema. These rules are affected by the data in the associated wagetypes. So a schema, and it's associated rules, needs a wagetype to direct it's functioning. These business objects are all interrelated.

In a table-driven business system application, such as an R/3 payroll business application, a wagetype business object (or its equivalent) may modify the functionality of a schema, and the data in the wageype can be configured by a user or whoever configures the payroll application. For example, consider a simple schema to calculate A and calculate B. The calculate B function and calculate A function are executable by driver source code **344** running the schema. But the order of execution of function A and function B, that is, whether to calculate A and then calculate B, or calculate B and then calculate A, is determined by the schema business object in configuration data **246**, which is consulted by payroll application source code **344** when executing. A user can modify the schema business object and thus modify the configuration of business application **340**. Source code may contain the executable statements implementing the algorithms of how to calculate A and how to calculate B, but the connection and the order is determined by the configuration data.

For another example, consider a rule business object that includes five calculation algorithms for a particular value. Configuration data, (e.g., in a wagetype object), may be set so that one algorithm is used for a specific group of employees at time X. At time Y, some legislation change or a union negotiation for the group of employees may take place and cause a need for a change for this particular group to switch to another calculation algorithm. To make the change, a user may modify a wagetype for this group employees in the business application by changing their processing class values so that they use another algorithm in their payroll calculations.

Common source code **370** represents code (and other business objects) that are stored in a common code base that business applications share among themselves. Consequently, in legacy business systems, including R/3, there may be no clear separation between the logical instances of execution or the logical instances of the data in a business application. Thus, when a user configures a part of a business application that is common to other applications, e.g., in common source code **370**, the user may affect many different instances of objects at execution. It is very difficult to verify that the changes made to the common code are correct for each and every business application that uses it, because they all must be tested to be sure.

In addition, configuration data **346** may be changed or created that causes the unintended and/or incorrect use of objects stored in common source code **370**. For example, in an SAP R/3 payroll business application, a user may configure a United States schema to use a Brazilian tax calculation function, which is accessible from common source code **370**. This is not correct, however, as the Brazilian tax function will perform a different calculation than a United States tax function.

As noted above, the data structure business objects, like wagetypes, and the functional business objects, like rules, are very interconnected and changes to one will typically affect the other. For example, changing part of a payroll application data structure object, such as a wagetype or a processing class of a wagetype, will likely affect the calculations for multiple employees done by multiple rules and related to multiple other wagetypes. And vice-versa. Consequently, in the table-based and table-organized form native to business system **330**, data structure business objects and functional business objects are difficult to understand, maintain, modify, debug, optimize, and correctly manipulate.

Some embodiments of systems and methods consistent with the invention represent business application objects as object-oriented programming language elements, building a model of the business application in some respects, and applying object-oriented programming language tools to the model. The model and the tools improve a user's ability to understand, maintain, modify, debug, optimize, and correctly manipulate the business application, as represented by the object-oriented programming language model.

In some embodiments consistent with the invention, anything that is related to the configuration of the business application may be considered a business object for transformation to an object-oriented programming language element(s). In one embodiment, a designer may select and define the business objects that are to be modeled as OOP objects. This selection and definition of business objects may be done using one or more different techniques, but generally the designer's goal should be to define business-application-layer objects, which are logical collections of things (data, operations, etc.) reflecting the design and functionality of the business application, and not objects based on the implementation or data structures that implement the business application, such as data-dictionary-layer structures. A business application typically uses a relational database model, but business object definition typically should not be based on the database relationships as they usually do not coincide with the logical business functionality.

In one technique, a designer may analyze the outputs of a business application and decide that the output data structures should be business objects in the business application model. For example, running a payroll application on an SAP R/3 system produces a table of result wagetypes, making wagetypes a clear candidate for a business object to be transformed into an OOP object. Smaller data structures that are part of larger data structures, such as processing classes, cumulations, and evaluation classes that are part of a wagetype, are also obvious candidates for transforming into OOP objects.

A designer may also analyze the business application to identify business entities that are treated as logical collections of data and/or associated operations by the business application, as these are also logical candidates for definition as business objections. For example, an SAP R/3 payroll application provides user interfaces to configure wagetypes, schemas, and rules to customize the application, treating them as logical entities. Thus, wagetypes, schemas, and rules may be considered business objects for modeling by OOP language elements. Logical business entities that encompass other logical entities may be decomposed to model as objects the encompassed logical entities that comprise them, such as decomposing wagetypes into processing classes, cumulations, and evaluation classes business objects.

After the business objects that comprise a model of a business application are defined, embodiments of systems and methods consistent with the invention may map them to specific object-oriented objects and elements. This may be done by analyzing the attributes of a business object and transforming them into an object-oriented language element, such as a Java™ language element that appropriately models the business object attributes. This process involves finding a corresponding construct, structure, or other element within the destination object-oriented language to represent each important attribute, element, or relationship of the business object being transformed. For example, an object-oriented language class, such as Java™ class, may be used as a starting point to represent most business objects. Business objects that are basically data structures, such as wagetypes in an R/3 payroll application, may be represented using static OOP elements in an OOP object such as a class, for example, as shown in FIG. **7**A (and explained further below). Business objects which contain executable code constructs, such as rules and schemas in an R/3 payroll application, may be represented using OOP objects having code, such as a class with methods or functions, for example, as shown in FIG. **25** (and explained below). Other object-oriented language features may also be used to model business object attributes. For example, object-oriented language superclasses and subclasses may be used to model the hierarchy and inheritancy attributes of business objects, and object-oriented language scope elements, such as public and private, may be used to model the scope attributes of business objects, etc.

This mapping process may be automated by defining a set of mapping rules used by a conversion engine to take input business object data for business objects and convert it into a corresponding object-oriented language object as described further below.

In addition to transforming business objects into corresponding object-oriented language objects, elements, etc., embodiments of systems and methods consistent with the invention may also use code-development-project-organization-tool features to model a business application. For example, as noted above, a legacy business system **330** typically shares common code and data structures **370** among business applications, so if something is changed in the common code, i.e., in the common parts, it affects every object or structure that uses that common part. One way to represent this feature of a business application in an object-oriented model may be to use code development project features and shared folders. For example, in many legacy table-based business systems like R/3, many business objects have a common configuration across runtime instances of the application, but each instance itself is independent, and different, at runtime. For a specific example, consider a business application used for a division of a business in Argentina and a division of the business in the United States. The application is configured using the same type of configuration tools for Argentina and the United States, but the running configurations of the application for each country are totally separate, and the same configuration objects within the context of the Argentine run of the application can mean different things than within the context of the United States run. They are like separate universes. Both universes, however, have a common foundation of business objects that are shared. These relationships and behaviors of the business application may be modeled using separate projects to hold, delimit, and manage the United States instances of the application objects and the Argentine instances of the application objects and by using shared folders to hold, delimit, and manage the common objects. Similarly, other OOP language development environment features, in addition to projects and shared folders, could also be used to model business application/business system features.

A designer modeling such aspects of a business system and business application may look at the relationships within the business system/application to identify "vertical dimensions" and "foundation" elements for projects and folders. A dimension may be thought of as an instance of all the objects involved in a process. A foundation element may be thought of as a common element that is used by many instances. So if an application uses the same structure more than once, but the data is different each time (i.e., there are many instances of the same structure), then these may be modeled as different dimensions (projects) with a common foundation (shared folder). Thus an object-oriented model of a business application/system organized by projects and shared folders enables

demonstration and evaluation of the effects to the dimension(s) of modifying a foundation element. Put in object-oriented terms, a foundation element may perhaps be analogized to a class definition for an object, while a vertical dimension may be analogized to an instance of the class object. Each instance may be the same architecturally, yet the data in each one is different. Such analogies and may be useful to a designer modeling complex table-driven business applications and systems having shared code and data structures and building transformation rules for converting business application objects and applications into object-oriented language representations.

Systems and methods consistent with the invention analyze a business application's data, operations, and relationships and create a corresponding object oriented programming (OOP) model corresponding to the business application entities. In one embodiment the OOP model may be realized in the Java programming language and may include OOP objects, functions, and operators. Each OOP model accurately represents the data, relationships, and rules associated with the corresponding business entity. Once converted to an OOP model, systems and methods consistent with the invention may use standard OOP tools to understand, manipulate, design, redesign, analyze, and modify the business application. This can be done independently of the native business application system, as the OOP model is separate and independent from the business application entities. Modifications made to the OOP model may then be converted back into business application entities and installed in the native system.

Detaching Application into Offline Mode

FIG. **4**A illustrates a flowchart illustrating an overview of an exemplary process **400** for creating an off-line model of a business application, consistent with the present invention. The exemplary process is applied to a business application composed in a first programming format. The programming format for such business applications are often specialized programming formats unique to each particular business application and, as described above, typically have limited flexibility in how a programmer may conduct efficient programming or debugging tasks.

As shown in FIG. **4**A, systems consistent with the invention may translate the business application into a model representing the business application (stage **410**). More particularly, the model reflects a representation of the business application in a second programming format different than the original or first programming format of the business application. In preferred embodiments of the invention, the second programming format may be an OOP language, such as Java or C++. The process for translating the business application into a model is described in greater detail below with respect to FIGS. **5** to **8**.

Once the system has generated the model of the business application, the system may then enable a user to process the model in a processing environment or platform corresponding to the second programming format (stage **420**). For example, as described above, the second programming format may be an OOP language, such as Java. Systems consistent with the invention may thus enable a user to program, debug, or analyze the programming code of the model by using an OOP editor or other type of OOP processing environment or platform. Because the user may process the model, as opposed to the actual business application, the user may do so while business application is still running. Thus, systems consistent with the invention allow a user to process a business application without interfering with normal business activities.

Further, by taking advantage of processing tools available with such OOP processing environments, the system may enable a user to use one or more tools to process the model. FIG. **4**B is an exemplary illustration **440** of the various processing tools consistent with the invention. As shown in FIG. **4**B, the system may enable a user to apply the processing tools to the model of the business application (stage **445**). These processing tools may include a deprecation process (stage **450**), a refactoring process (stage **455**), a runtime error process (stage **460**), a development process (stage **465**), an optimization process (stage **470**), a design tools process (stage **475**), and a version compare process (stage **480**). These and other processing tools consistent with the invention are described in greater detail below with respect to FIGS. **9** to **15**. Further, FIGS. **17** to **56** illustrate exemplary user interface screens enabling a user to process the model representing the business application.

The system may then re-translate the model representing the business application in the second programming format back to the first programming format (stage **430**). As described below, systems consistent with the invention may translate the model in such a way that any modifications made to the model in the second programming format are automatically reflected in the re-translated business application composed in the first programming format. The process for re-translating the model back to the first programming format is essentially the reverse of the forward translation process described above with respect to stage **410**, and as illustrated below with respect to FIGS. **5** to **8**.

FIG. **5** illustrates the exemplary modeling process **500** of FIG. **4** in greater detail. As shown in FIG. **5**, to translate the business application into the second programming format, the system receives the logic entities of the business application composed in the first programming format (stage **510**). As described above with respect to FIG. **1**, these logic entities may be the business data structures (e.g., configuration data **100** and business rules **125**) of the business application. Based on the received logic entities of the business application, the system may then generate a model representing the business application in the second programming format (stage **520**). As also describe above, the second programming format may be, for example, an OOP language such as C++ or Java. The system may then, as described above with respect to FIG. **4**, process the generated model of the business application (stage **530**). The system may then re-translate the processed model to regenerate the business application in the first programming format (stage **540**).

Generating a Model of a Business Application

FIG. **6**A illustrates an exemplary process **600**, consistent with the invention, for translating the business application into a model composed in the second programming format. As shown in FIG. **6**A, the system may receive the data structures, which may be table-based data structures, of the business application (stage **610**). In this regard, the system may scan the business application to identify the table-based data structures, which may be any group or segment of data reflecting data stored in a table-based format of the business application. The table-based data structures may, for example, correspond to the configuration data and rules of the business application. Further, each table-based data structure may include attributes of the data stored in the respective table-based data structure. For instance, as explained above with respect to FIG. **1**, a table-based data structure may

include configuration data **100** having attributes **105** to **120** or a rule **125** having operation attributes **130** to **140**.

The system may then select the minimum set of attributes of table-based data structures that uniquely define the table-based data structures (stage **620**). To this end, the system may first identify those attributes of, for example, the configuration data (e.g., attributes **105** to **120**) and any rules (e.g., operation attributes **130** to **140**). The system may then identify those attributes that are common to each table-based data structure. From the identified common attributes, the system may then select the least number of these common attributes that uniquely defines each translational data structure.

Based on the selected minimum set of attributes, the system may then define one or more translational data structure(s) (stage **630**). In systems consistent with the invention, a translational data structure may be an identified data or business structure of the business application that is converted into an object-oriented structure. For example, in an R/3 payroll application, the minimum set of attributes may include a geographic attribute (e.g., "New Zealand"), a version reference attribute (e.g., "rna"), and a client reference attribute (e.g., "800"). The corresponding translational data structure may thus be defined as NewZealand.rna.800. In OOP terms, the defined translational data structures may include at least one translational data structure that respectively corresponds to a project, an object, and an object instance. The system may then generate the model of the business application based on the defined translational data structures (stage **640**).

FIG. **6**B illustrates an exemplary process **650** further describing the processing stages of FIG. **6**A. As shown in FIG. **6**B, the system may receive table-based data structures corresponding to configuration data and rules of business application (stage **660**). The system may then identify the attributes of received configuration data (stage **665**) and identify the attributes of received rules (stage **670**). The system may then determine those attributes common to the identified configuration data and to the rules of the business application (stage **675**). From these common attributes, the system may then select the minimum set of common attributes that uniquely define a translational data structure (stage **680**).

As noted above, methods and systems consistent with the invention may transform business application logic entities or structures into OOP structures that represent the data, relationships, and operations of the business application structures. In one embodiment, the business application structures are transformed at the application level, including consideration of application logic, as opposed to the database table level. Thus, a complex business structure comprising ten or more related database tables, such as a wagetype in an SAP R/3 payroll processing application, may be transformed into a single OOP structure, such as a Java class, that encompasses all the information in the ten or more related database tables.

In some embodiments consistent with the invention, business structures are transformed into a hierarchical object-oriented class structure. For example, as described above, all the related business structures in a business application may be analyzed to identify common elements or attributes, and transformation involves defining an OOP superclass that represents the common elements. Then, specific business structures are analyzed to identify their unique elements, and each specific business structure is transformed into an OOP structure by defining subclass instances of the superclass, which add representations of the unique elements of each specific business structure to the superclass. This exposes the hierarchy of the business structures to a user or administer viewing the OOP structures, and allows manipulation of the OOP representations of the business structures by OOP design and development tools. In one embodiment, included in the transformation to OOP subclasses is extraction of the data from the business structure for inclusion in the subclass. Thus, the values of various fields are also exposed to users and OOP tools.

Processes consistent with the invention for generating an OOP model of a business application are described in further detail below with respect to FIGS. **7** and **8**. FIG. **7**A, for example, illustrates an exemplary OOP structure consistent with the invention. Referring briefly back to FIG. **2**A, recall the example of a wagetype business structure **200** from a payroll application of an SAP R/3 system, which may have several attributes **205**, such as several processing class attributes. An implementation consistent with the invention may transform the data from a specific wagetype business structure **200**, for example the wagetype structure known as "/123," into an OOP structure such as a subclass that extends a "wagetype" superclass as shown in FIG. **7**A. The transformation process creates a valid OOP language name "wagetype_s123" **792** based on the business structure name "/123" and declares it as a subclass of the superclass wagetype **794**. The OOP structure corresponding to business application wagetype /123 contains and models the data and relationships **796** of the business application wagetype /123 extracted from the database tables of the business application. In the example shown, included in the OOP structure are code lines that model the processing class attributes **798** of the business application structure wagetype /123. Also included in the example shown is documentation **795** related to the business application structure wagetype /123, which may be used and displayed by OOP tools such as JavaDoc™.

In some embodiments, transformation may be based on a set of rules mapping or defining what OOP structure may create or define each business structure encountered in the business application. In one embodiment, the set of transformation rules may be defined manually. The rules for translating a business application structure into an OOP structure may create syntactically correct OOP structures (e.g., in Java™ code) useful for processing at the business application level. For example, one transformation rule for mapping various business structures into OOP structures may be to group them according to whether they share operations that can be performed on them. Another rule may be to create a subclass for each different time period attribute of a business structure based on start date and end date value of the time period attribute. Another rule may be to, within a subclass, define each business structure configuration element, such as processing class attribute, as an OOP operation in that class. Yet other rules may define when to use a public, private, or shared OOP class attributes to determine the scope and visibility of each OOP structure. Yet another rule may be to take each business structure as defined by the business application and create a corresponding OOP structure, effecting a one-to-one conversion mapping.

FIG. **7**B illustrates an exemplary process consistent with the invention for translating, for example, configuration data. As shown, the process begins by defining transformation rules for converting table-based data structures of a business application into object-oriented structures (stage **710**). Next, the process scans the business application to identify a table-based data structure (stage **720**). Next, the process converts the identified table-based data structure into an object-oriented structure, according to the transformation rules (stage **730**).

The process then determines whether all the table-based data structures in the business application have been con-

verted into object-oriented structures (stage **740**). If not (stage **740**, No), the process loops up and continues to identify and convert. If so (stage **740**, Yes), then the process expresses the object-oriented structures in an object-oriented programming construct, such as an OOP class (e.g., a Java™ class) that represents the table-based data structure.

FIG. **7**C is a block diagram of an exemplary architecture, consistent with an embodiment of the invention, for transforming a business application. As shown, a business application **755**, such as the Payroll application in an SAP R/3 system, includes business application structures **757**, which are logical entities embodied in various database tables or other data formats. In one embodiment, business application **755** is a preexisting application (such as the an application in an SAP R/3 system) that has been modified by adding remote function calls **760** that retrieve business application structures at the request of an outside application.

In the embodiment shown, extract module **765** implements a process that retrieves business application structure data from the business application and readies the data for transformation into an object-oriented structure. For example, extract module **765** may send a data request via remote function call **760** to business application **755** and receive, in response, all the appropriate application data about a business structure **757**, including context data about business structure **757**. The application data from business application **755** may also include information about the data, e.g., documentation describing the data. In the embodiment shown, extract module **765** maintains a context directory **770** containing information regarding what context is needed for each activity and a translation module **775** that translates the business application data into XML, or a similar markup language.

Transform module **780** receives the XML-formatted business application data from extract module **765**. Transform module **780** contains an embedded conversion mapping component **785** that is used in transforming the XML-formatted business application data into an object-oriented structure. Conversion mapping component **785** includes transformation rules, as described above. For example, a business application structure logic entity, such as a "standard deduction" application structure (which may consist, in business application **755**, of a number of physical database tables containing various content data), may be mapped with a conversion algorithm into a Java™ class or a set of Java classes. In one embodiment, the name of the resulting object-oriented structure may typically correspond to the name of the logic entity from business application **755**. For example, the model representing business application **755** may include a "standard deduction" Java™ class, and instances of the class representing each type of standard deduction in business application **755**. Documentation information describing the business application structure from business application **755** may be transformed into comments, such as Java™ language source code comments, or the like and associated with the OOP construct created to represent the business application structure.

Transform module **780** provides the object-oriented structure representing a business application structure to an object-oriented language tool **790**, such as a Java™ integrated development environment tool, a Java™ code optimizer, a configuration management tool, etc or an integrated development environment, such as Eclipse™ (www.eclipse.org), that incorporates many software development tools. Object-oriented language tool **790** may be used to understand, maintain, develop, design, redesign, modify, configure, or otherwise manipulate the OOP model of business application **755**. For example, a customized wagetype editor/viewer may be included as part of an integrated development environment for an SAP R/3 system, presenting object-oriented structures such as the class shown in FIG. **7**A on easy to understand, use, and navigate user interfaces, such as the examples shown in FIGS. **22** and **23**.

In one embodiment consistent with the invention, extract module **765**, transform module **780**, and object-oriented language tool **790** may be implemented on a computer(s) separate from the computer implementing business application **755**, such that business application **755** can be modeled, and the OOP model manipulated, without interfering with the ongoing operation of business application **755**. Further, one of ordinary skill will recognize that Java™ is merely used as one example of an OOP language, and that the scope of the invention includes other OOP languages, such as C++, C#, Visual Basic, etc.

As with data structures (e.g., for configuration data) in legacy business applications, the operations, instructions, or business rules that implement the application are also typically table-based. That is, the operations that perform a business rule function, such as deducting payroll taxes from an employee's paycheck, (and some of the relationships between the operations), may be contained in database tables. The operations may be executed by application source code that traverses the tables when carrying out a particular rule. And, similar to business data structures, business rules or operations may be contained in many different database tables, which entail the same drawbacks described above for table-based business data structures. Perhaps worse, there may be no physical connection between the rules tables and the data tables they operate on and vice-versa. Just as with table-based business application structures, these table-centric rules and operations are difficult for users and administrators to comprehend, manage, create, revise, maintain, and debug.

Embodiments of methods and systems consistent with the invention transform business application operations, rules, and schemas into OOP constructs that represent the operation, rule, or schema, emulate its operation in terms of sequence with other operations, and represent its relationship(s) to the data it operates with and upon. A rule may be thought of as a sequence of operations that work on data structures to perform a business function. A schema may be thought of as a business process or workflow that connects a sequence of rules and data structures to perform a more complex business function, for example, as illustrated in FIG. **2**C.

In one embodiment, an automated system determines each operation that makes up a schema or rule in the business application by analyzing the database table(s) that hold the schema or rule, for example, by analyzing the rule tables in an application in an SAP R/3 system or other business application. In one embodiment consistent with the invention, for each business operation found, the system transforms the operation into part of an OOP model by declaring or creating an empty OOP language construct with a similar name to that of the business operation, such as a Java™ function or method. The system may insert a standard number of parameters, or make other modifications, to make the syntax of the newly declared OOP construct valid for the programming format being used, such as the Java™ language. The system preferably puts the OOP constructs in the same sequence as the business application operations to form an OOP model object representing the rule, schema, etc. that encompasses the business application operations. Thus, in some embodiments consistent with the invention, the system produces a model that represents the configuration of the business appli-

cation rules and schema, but not their functionality, as the OOP constructs are null functions.

In one embodiment, the OOP language code representing the business operations may contain function or method calls only, while the bodies of the functions or methods are empty. OOP languages such as Java™ allow a null function body and can compile a function call with a null function body. For example, a payroll business application for an SAP R/3 system may contain a business rule with the operation "NUM=." An embodiment of a system consistent with the invention may create a syntactically correct null Java function named "NUM=" when modeling the payroll business application. Thus, as noted, the system may produce a model that represents the configuration of the business application rules and schema, but not their functionality, as the OOP constructs are null functions. Since the OOP language syntax is correct, OOP language tools, such as compilers, analyzers, optimizers, etc. can be used to find errors or problems in the configuration model, such as errors in relationships, calls, interactions, design, etc. associated with the modeled null functions. This is useful because any errors associated with the model are also associated with the business rules and operations it represents. For example, if a modeled OOP function calls another OOP function that does not exist (because there is no corresponding business application operation that caused the called OOP function to be created in the model), this error will be exposed by the OOP language tools, such as a compiler or linker.

In some embodiments consistent with the invention, the OOP functions include documentation pulled from the business application to help users understand the function. In some embodiments consistent with the invention, the process that transforms business application operations or rules into OOP constructs that represent the operation or rule is controlled by a set of transformation rules. These transformation rules may be generated manually. The transformation rules should be designed such that the resulting OOP language function enforces the scope and other characteristics of the business application operation through OOP language techniques. For example, a business application rule (and therefore the operations that comprise it) may be valid for only one country, or in a few countries, or in all countries. To enforce this scope, the transformation rules may produce functions of appropriate scope, such as global, shared, or local. In some embodiments, an OOP project model may also be used to enforce the scope of OOP objects and functions.

In a manner similar to that explained above regard transforming business application data structures into OOP structures, the exemplary architecture shown in FIG. **7**C may be used to implement a system that also transforms a sequence of business application operations (e.g., business application rules or schemas), into object-oriented language constructs. In the architecture shown, object-oriented tool **790** may include a customized rule editor/viewer (not shown) as part of an integrated development environment for an SAP R/3 system, presenting easy to understand, use, and navigate user interfaces, such as the examples shown in FIGS. **17** and **25**. As explained further elsewhere in this application, the OOP constructs representing the business application rules and schemas may be modified, changed, optimized, etc. using OOP language tools, and then reverse transformed back to the business application format and the run in the business application.

As with data structures (e.g., configuration data) and operations (e.g., rules) in business applications, the relationships or links between business objects, such as business data and business operations, are also typically table-based, making the relationships difficult, at best, to discern statically. For example, as illustrated in FIG. **3**, the relationships between configuration data **305** and business rules **310** may not be realized until runtime. This makes it difficult for users and administrators to detect errors in the relationships, such as missing business data that is referred to by other data or a business rule, especially when working with the tables at any time other than run time.

Methods and systems consistent with the invention transform business application relationships or links into OOP constructs that represent the relationship, tying together different types of business objects, such as data and operations, in an easily comprehensible and manageable manner, and in a manner that allows automated OOP tools to check the integrity of the relationships. In one embodiment, an automated system uses an object-oriented language construct, such as an assignment operator (e.g., "=" in Java), to link together OOP objects representing business data and business operations. The object-oriented language construct binds different OOP objects representing business application objects and shows and enforces a relationship between them. For example, as shown in the exemplary implementation of FIG. **7**A, the processing class objects **798** of a wagetype object may be related to processing class value objects, and these relationship are represented by the Java assignment operator "=." The processing class object "WageType_s123.ProcessingClass03," for instance, is linked to the value of the business object "processingclasses.ProcessingClass03. Value0" by the equality operator "=."

Other business objects may also be linked. For example, a business rule object may be linked to a wagetype object that it affects. For instance, as shown in the example of FIG. **27**, a variable object in a rule may be linked to a wagetype processing class attribute, with that link represented by an assignment operator.

Representing the business object relationships as OOP language constructs, such as an assignment operator like "=," allows the use of a OOP language tool, such as a Java debugger, to walk through referential relationships and discover referential problems in the integrity of the business application layer. For example, as shown in FIG. **7**A, if the value of processingclasses.ProcessingClass03.Value0 is not defined, then the compiler will flag a reference error when it processes the equality operator in the statement "WageType_s123.ProcessingClass03=processingclasses. ProcessingClass03.Value0." Thus, when trying to resolve the assignment operator relationship, a Java compiler will be able to identify broken links or references, which represent empty links in the corresponding modeled business application. Consequently, for an instance of wagetype, for example, a user can see from the Java code model what values are acceptable for each processing class. The model also lists exactly which processing classes are part of the wagetype, and thus any unlisted processing classes are not defined for that wagetype (and so not allowed). The assignment operator and processing classes included in the wagetype OOP structure are examples of integrity checks for the business application.

FIG. **8** illustrates an exemplary process consistent with the invention. As shown, the process may begin by defining transformation rules for converting logical connections from a business application into object-oriented structures that represent the logical connections (stage **810**). Next, the process may scan a business application to identify a logical connection or link between table-based data structures or between table-based data structures and rules (stage **820**). Next, the process may convert the identified logical connection or link into an object-oriented structure, according to the transfor-

mation rules (stage **830**). In one embodiment, the transformation rules include using the equality operator of an OOP language as the object-oriented structure. In some embodiments, the transformation rules are defined manually according to criteria similar to that described for transforming business structures and business rules.

The process may then determine whether all the logical connections or links in the business application have been converted into object-oriented structures (stage **840**). If not (stage **840**, No), the process may loop back and continue to identify and convert logical connections. If so (stage **840**, Yes), then the process may display the object-oriented structures as an object-oriented programming construct, such as an assignment operator, between related business data and/or business operation representations (stage **850**).

In the exemplary embodiment shown, the process may then identify any problems associated with the object-oriented structure, such as an unresolved reference or other integrity problem (stage **860**). Such problems may be identified by convention OOP language tools, such as compilers and linkers. Any such problem represents a problem in the corresponding business application structures and relationships, and the OOP language constructs model the relationships and behaviors of the business application from which they are derived.

As with data structures (e.g., configuration data) and operations (e.g., rules) in business applications, the relationships or links between the business data and business operations are typically table-based, making the relationships difficult, at best, to discern with the table-centric views and tools provided by a typical business application system. For example, as illustrated in FIG. **3**, the relationships between configuration data **305** and business rules **310** may not be realized until runtime, making it difficult for users and administrators to comprehend, manage, create, revise, and debug such relationships, especially by working with the static tables before runtime. In another example, a single business rule may be stored in multiple tables. To determine all of the configuration data used by such a business rule, it may be necessary to study all of the multiple tables storing the rule. This process can be time-consuming and error-prone, just to understand the relationship between a business rule and the data on which it operates.

Embodiments of methods and systems consistent with the present invention transform business application objects and relationships into object-oriented programming constructs, making it possible to display business objects and their relationships in an easily comprehensible and manageable manner.

Using processes described above, an object-oriented model may be generated to represent a table-based business application. The object-oriented model may represent logical entities of the business application, such as rules, configuration data, and schema. The model may also represent links between the logical entities. In this way, object-oriented tools and graphical user interfaces may be used to display the links between the logical entities of the business application.

For example, as shown in FIG. **31**, a user may select a business object such as an evaluation class, e.g., evaluation class 02, "Wage type assignment for pay statement." This business object may be chosen from a menu of available evaluation classes, as shown in the upper center right panel with the tab labeled "Evaluation Class." By processing the object-oriented model of the business application using common object-oriented tools and methods, all business application objects that reference evaluation class 02 may be determined and displayed. For example, the referencing objects may be displayed in a list of References, as shown in the bottom right hand pane with the tab labeled "References." A total number of referencing objects may be calculated and displayed, such as "(found 785)" in the bottom right hand pane with the tab labeled "References."

Thus, a user may easily see all the business objects that depend upon or are somehow related to the selected business object and gain awareness of the scope of objects in the business application that may be affected by a change to the selected business object. By using an object-oriented model of the complex, table-based business application, relationships not previously understandable may now be easily determined and displayed.

In another embodiment consistent with the invention, an outline may be generated and displayed to expose the structure of business objects within a business application For example, in FIG. **17**, a single integrated display shows information pulled from an R/3 system in a Javadoc displayed in the lower left pane. This information includes, for example, a name of a business application object (e.g., payroll.schemas), a use of the business application object (e.g., "This function retrieves employee health plan benefit information . . . "), business application syntax for the object (e.g., Fkt Par1 Par2 Par3 Par4), and an explanation of the parameters.

This information may be displayed simultaneously with an object-oriented model of the business object. In the lower right pane of the single integrated display shown in FIG. **17**, pseudo-Java code is displayed that represents payroll.schemas as an object-oriented package. Drop-down lists and pop-up windows may be displayed with the integrated display to provide more information. For example, the information from R/3 shown in the lower left-hand pane may also be shown in a pop-up window over the object-oriented pseudo-code when a user selects a certain part of the object-oriented structure.

In these ways, methods and systems consistent with the present invention use object-oriented models and tools to determine and display the relationships between logical entities in a table-driven business application.

Processing a Model of a Business Application

The following sections describe exemplary embodiments for processing a model of a business application. While the description below provides headings

Exposing Deprecated Entities

Business applications often involve large, complex legacy systems. Over time, these systems evolve to accommodate changing business rules, user needs, etc. Each change to a business application may affect many logic entities within the business application. For example, changing a processing rule may result in another logic entity, such as configuration data, that is no longer used by the business application. However, because the business application is large and complex, the user making the change may be unaware that the logic entity is now unused and could be deleted. Instead, the unused logic entity remains in the business application even though it is never again used.

As time passes, a business application may develop a great deal of unused, or "deprecated" logic entities, including configuration data and processing rules. Traditional table-based business applications do not have a mechanism for identifying or removing deprecated logic entities. This results in wasted storage space and unnecessary programming complexity associated with conventional business applications.

Using methods and systems consistent with the present invention, deprecated logic entities of a business application may be identified using an object-oriented model of the business application. By transforming business application struc-

tures into object-oriented structures, deprecated logic entities in the business application may be exposed using object-oriented referential integrity checking.

For example, an object-oriented code compiler checks syntax and identifies potential problems, such as mixed data types, broken references, etc. Because a compiler does not actually run the code, it can be used to analyze an object-oriented model consistent with the present invention. The potential problems identified by the compiler in the object-oriented model may correlate to deprecated logic entities in the business application. These potential problems may be marked in an integrated configuration environment to indicate deprecated logic entities that could be removed to streamline the corresponding business application.

Each logic entity in a business application may be transformed into a corresponding object-oriented structure. An object-oriented compiler may then be used to check the referential integrity of the object-oriented structures. Although the object-oriented compiler thinks it is finding problems, such as broken references, in object-oriented code, it is actually identifying problems in the underlying business application. For example, the object-oriented compiler may identify a reference to a branch of code that will never be executed. By correlating the identified branch of code with its corresponding logic entity in the business application, a deprecated logic entity is identified. In one embodiment, deprecated logic entities may be removed from the business application after they have been identified using the object-oriented model.

In one example, a business application logic entity, such as "wagetype," may not be defined in the business application for a certain country such as New Zealand. However, "wagetype" cannot be deleted from the business application altogether because it is defined for other countries. During transformation, the object-oriented structure corresponding to wagetype in New Zealand may be created but marked as deprecated to indicate that it is unused in the underlying business application. Deprecated structures may be marked, for example, using yellow underline in displayed object-oriented programming language. Yellow might be used because the deprecated structure is not an error that needs to be fixed (which might be shown in red). In this way, a user viewing the object-oriented structure is alerted to deprecated structures in the business application and can more readily study the effects of changes.

FIG. **9** is a flowchart of a method for exposing deprecated business application logic entities using an object-oriented model. In this embodiment of the present invention, a model representing logic entities of the business application is received (step **910**). As discussed above, the model may be, for example, in an OOP format. The model is processed using a code compiler, such as an object-oriented code compiler, to identify a possible exception in the model (step **920**). The possible exception in the model is correlated to an unused logic entity in the business application (step **930**), and an indicator of the possible exception may be displayed with the model to depict the unused logic entity in the business application (step **940**).

For example, a code compiler may identify a dead pointer in an object-oriented model of business application logic entities. The dead pointer may correlate to an obsolete rule (e.g., a repealed or expired tax) in the business application. When the model is displayed in an integrated configuration environment, the corresponding object-oriented code structure may be underlined or highlighted or otherwise marked to indicate the deprecated business application logic entity. Other possible exceptions include orphans, cost structures that do not exist, outdated regulations, expired taxes, etc.

One example of a display consistent with the present invention may be found in the screen shot shown in FIG. **25**. A business application rule, "W020-Cumulation of gross amount," is displayed using an object-oriented model in a large window of a single integrated configuration display. To the right side of the large window is a series of rectangular flag markers indicating possible problems identified by a code compiler applied to the object-oriented model. When the cursor is placed over a flag marker, a pop-up box may be displayed reading, for example, "RuleWRFI cannot be resolved." This may indicate to a user that the business application logic entity corresponding to the object-oriented model may be deprecated.

Refactoring

Modifications made to a business application can make an already complex system even more unwieldy to understand and maintain. These difficulties are compounded by the fact that table-based business applications do not allow the use of design techniques available in other types of systems. Refactoring is a programming technique for improving the design of existing software code without altering the code's behavior. Refactoring may improve code by, for example, consolidating many statements into one, decomposing long methods into shorter ones, eliminating redundancies, etc. Refactored code may be easier to read and understand, simpler to debug, and more efficient to run and maintain.

Systems and methods consistent with the present invention enable the use of refactoring to reorganize the configuration of a business application. By transforming business application structures into object-oriented structures, object-oriented refactoring tools may be used to retool the underlying business application.

Program refactoring tools may be applied to object-oriented structures to edit the structures without affecting their behavior. Refactoring may be a recursive process, taking several passes through a structure or code section in order to optimize and improve it. Refactoring might be performed on an ongoing basis to maintain code and structures, or it might be applied for a specific purpose, such as preparing an application for a major upgrade.

Using this invention, any number of object-oriented refactoring tools may be applied to the business application through application to an object-oriented model of the business application. These tools might include:

(1) Rename refactoring: This function enables the renaming of any package, class, method or variable, and automatically finds and corrects all references to it. By applying the rename tool to the object-oriented model, the tables and references in the underlying business application may be better organized or named.

(2) Extract method refactoring: This function analyzes a selected piece of code and transfers it into a separate method in order to aid program modularization and break code into manageable pieces. By applying the extract method tool to the object-oriented model, the underlying business application may be better organized and easier for users to understand.

(3) Inline variable refactoring: This feature replaces all references of a variable with a declaration expression, then deletes the older variable text. This can save time and effort when rearranging code or introducing upgrades, such as new tax laws or new employees. By applying the inline variable tool, the business application may be updated with a minimum of potential referencing errors.

(4) Clean imports refactoring: This function allows a user to scan code in order to find and remove unnecessary import

statements. Applying this tool may assist with identifying and cleaning up superfluous elements in the underlying business application.

A skilled artisan will recognize that these and may other refactoring tools and methodologies may be applied to the object-oriented structures in order to effectively refactor the corresponding business application.

Using methods and systems consistent with the present invention, each logic entity in a business application may be transformed into a corresponding object-oriented structure. An object-oriented refactoring tool may then be used to refactor the resulting object-oriented structures. The refactoring tool will improve the object-oriented structures by, for example, consolidating many statements into one, decomposing long methods into shorter ones, eliminating redundancies, etc. However, the operation of the code will not be affected. Once refactoring of the object-oriented code is completed, the object-oriented code may be translated back into its corresponding business application logic structures. In this way, the business application will reflect the improvements made by refactoring the object-oriented code.

FIG. **10** is a flowchart showing an exemplary method for redesigning a business application composed in a first programming format consistent with an embodiment of the present invention. When a model representing the business application in a second programming format is received (step **1010**), the model may be processed using a code refactoring tool in the second programming format (step **1020**). In one example, the first programming format may be a table-based format, such as R/3, and the second programming format may be an object-oriented programming format, such as JAVA™.

Modifications to the model made or suggested by the code refactoring tool may then be analyzed (step **1030**). Modifications suggested by the code refactoring tool may include, for example, removing redundant code, renaming variables or methods, breaking large routines into shorter ones, etc. Each modification may be analyzed before it is accepted, or modifications may be party or fully automated. After the modifications are accepted and/or applied to the object oriented model, the object-oriented code may be translated back into its corresponding business application logic structures to propagate the changes into the business application, thus effectively refactoring the business application itself. In this way, refactoring modifications may be applied, in the first programming format, to the business application (step **1040**).

In one example, the code refactoring tool includes removing redundant code of the model in order to remove redundant code in the corresponding business application. In another example, the code refactoring tool includes consolidating many statements into one, which may in turn streamline the corresponding business application. In yet another example, the code refactoring tool includes decomposing long methods into shorter ones, making the corresponding business application easier to understand and work with.

Runtime Errors

Given the complexity of a business application, each change made to it may affect many logic entities within the business application, resulting in errors that may not be apparent until runtime. For example, changing configuration data related to a project may result in a call to a processing rule that does not exist for that project. However, because the business application is large and complex, the user making the change may be unaware that the call will result in an error during configuration. Instead, the user may not realize the error until runtime, when it is costly and time-consuming to correct. Traditional table-based business applications do not have any simple mechanism for identifying or removing runtime errors at design time.

Using methods and systems consistent with the present invention, runtime errors in a business application may be identified prior to runtime using an object-oriented model of the business application. By transforming business application structures into object-oriented structures, runtime errors (such as unresolved references) in the business application may be exposed using object-oriented referential integrity checking.

Each logic entity in a business application may be transformed into a corresponding object-oriented structure. An object-oriented compiler or debugger may then be used to identify errors, such as syntax errors or broken references, in the object-oriented structures. Although the object-oriented compiler thinks it is finding problems in object-oriented code, it is actually identifying problems in the underlying business application. For example, the object-oriented debugger may identify a reference to a branch of code that does not exist. By correlating the identified branch of code with its corresponding logic entity in the business application, a problem with dependencies in between business application structures may be identified. In one embodiment, a user may correct the problems in the object-oriented code and translate the corrections back to the business application before running the business application.

A runtime error in the business application may be, for example, calling a rule that does not exist for a project, such as a country, or attempting to create a wagetype that does not exist for a project, such as a country. In one embodiment, when business application structures are transformed into object-oriented structures, a referential error might be revealed by a class that is declared but empty. These errors, once identified using the object-oriented model, may be displayed using, for example, red underline to show that runtime errors will occur in the business application. The identifier may be propagated up through a number of user interfaces to show potential problems at any level of the system.

A sample screen shot of how potential errors may be displayed is included in FIG. **29**. In the integrated configuration environment of FIG. **29**, a business application rule, "W020—Cumulation of gross amount," is displayed for project USA.rna.800 using an object-oriented model in a large window. To the right side of the large window is a series of rectangular flag markers indicating possible problems identified by a code debugger applied to the object-oriented model. When the cursor is placed over a flag marker, a pop-up box may be displayed reading, for example, "RuleWRFI cannot be resolved." This may indicate to a user that the business application logic entity corresponding to the object-oriented model has a potential runtime error. The display also includes a list of problems (e.g., "Rule 9112 cannot be resolved") in the lower right portion. The problems may be further displayed up the hierarchy of menus using, e.g., a red "X" to show the problem in the list of Payroll Rules, the list of logic entities within project USA.rna.800, and the list of all projects. By propagating the error flag up the menu hierarchy, users may be alerted of many potential runtime errors quickly and effectively.

In traditional table-based business applications, a log file may be created to trace the path of processing during a sample run (e.g., a payroll run) of the business application. The log file may note errors present in the business application. However, the log file is typically long and difficult to debug. By using object-oriented programming constructs, debugging is greatly simplified. The sequential view of a log file is replaced

by an object-oriented data representation view that is familiar to developers today. In this way, the same data used for debugging may be presented in a much more user-friendly fashion. In one embodiment, the debugging may be performed offline, increasing efficiency and flexibility of maintaining the business application.

FIG. **11** is a flowchart for a process for analyzing referential integrity of a business application consistent with embodiments of the present invention. A model representing the business application may be received (step **1110**) and processed using a code compiler to identify a compiler exception (step **1120**). The compiler exception may be, for example, a broken reference, an infinite loop, etc. The compiler exception in the model may be correlated to a referential integrity instance in the business application (step **1130**), and correction of the compiler exception in the model may be enabled (step **1140**). Correction may be enabled through displaying flags or other indicators to call the problems to the user's attention and providing configuration tools via an integrated configuration environment. Finally, the correction to the business application is applied to correct the correlated referential instance (step **1150**).

One skilled in the art will recognize that many different object-oriented code analyzers, including debuggers, interpreters, compilers, etc. may be used to identify possible runtime errors in the business application and that many different ways of marking and correcting these errors may be employed consistent with embodiments of the present invention.

Interactive Development Environment (IDE)

Business applications, such as the R/3 system created by SAP, often involve large, table-based systems and evolve over many years as businesses grow and change. Highly trained consultants may be employed to implement, configure, and maintain a business application for a company. Due to the complexity of traditional business applications, it may take years to configure an application for a single company. Because of their legacy nature and highly customized usage, business applications typically have limited development tools to assist in the configuration and maintenance of the applications.

To assist in the configuration and management of such business applications, systems and methods consistent with the present invention enable business application consultants to configure and maintain business applications using an interactive configuration environment that offers the simplicity and usability of an IDE.

An IDE is a set of tools available to assist a software developer in writing and maintaining software code. IDEs enable users to design and manipulate code, such as object-oriented code, in an easy-to-understand manner. An IDE may include multiple programs that are run from a single user interface. For example, programming languages often include a text editor, compiler and debugger, which are all activated and function from a common menu. Other IDE tools may include a version control system, GUI design tools, a class browser, an object inspector, and a class hierarchy diagram. IDEs are available for specific programming languages, e.g., the Visual Basic IDE, or for multiple languages, e.g., the Eclipse IDE.

An interactive configuration environment consistent with the present invention integrates and displays a table-based business application using a single, interactive display. For example, users can write a new business rule using familiar object-oriented code, and that code may be converted into logic entities to apply the new rule in the business application. In another example, users may view existing business applications using an IDE interface. In this way, users may not need to worry about the format or structure of the business application but may still be able to understand and configure it.

When business application structures are displayed using object-oriented code structures, the business meaning of the object-oriented code structures may also be displayed in the integrated environment. For example, when a user rolls a mouse over an item in the object-oriented code, documentation of the underlying business meaning may be displayed in a pop-up box. This aids the user in understanding a complex business application using familiar, easy-to-understand object-oriented programming constructs.

Using an object-oriented user interface, the user may browse business application logic entities and related objects using a single integrated display. Object-oriented code assists and templates may be available to assist users in writing new object-oriented code in order to add new logic entities to the business application. A template may consult a translation key to determine syntax, parameters, etc. and display these options in a drop-down list in the object-oriented code display. Using IDE tools, a state of the business application may be compared with a changed state of the business application, for example, by using a side-by-side display.

FIG. **12** is an exemplary flowchart of a method for analyzing the configuration of a business application consistent with embodiments of the present invention. When a model representing the business application is received (step **1210**), the model may be displayed using a developer interface (step **1220**). The model and corresponding business application may be integrated and displayed using a single, dynamic display, e.g., the integrated configuration environment shown in FIG. **17**. The integrated configuration environment may include, for example, a series of windows displaying business application entities as a hierarchy of projects, rules, and schema. The environment may also include an object-oriented model of a business application object displayed together with documentation pulled from the business application.

When a change to the model is received through the developer interface (step **1230**), the change to the model may be correlated to a corresponding configuration change in the business application (step **1240**). The corresponding configuration change may be applied to the business application or it may simply be used to model possible changes.

For example, a consultant may wish to modify a rule in the business application. Rather than analyzing the table-based structure of the business application to determine the tables that hold different parts of the rule, the consultant may use the integrated configuration environment to edit the rule using its object-oriented model counterpart. As part of the integrated configuration environment, the proper syntax or available parameters may be automatically displayed to assist the consultant in making his edits. Once the edit is complete in the object-oriented model, the change may be translated back to the underlying business application and automatically applied, e.g., to the many tables that hold different parts of the rule.

In another example, a consultant may wish merely to determine the effects that a potential change to the business application would have. Using an object-oriented model of the business application, the consultant could try out the potential change by applying it in the integrated configuration environment to see what the effects would be if the change were committed back to the business application.

A skilled artisan will appreciate that many other configuration tools would be made available to users of a business

application by displaying and manipulating an object-oriented model of the business application using an integrated configuration environment consistent with the present invention.

Checking Validity of Business Applications

As described above, business applications often involve large, complex legacy systems. Over time, these systems evolve to accommodate changing business rules, user needs, etc. Each change to a business application may affect many logic entities within the business application. Furthermore, changes to the business application can make an already complex system even more difficult to maintain. These difficulties are compounded by the fact that table-based business applications do not allow the use of design and optimization techniques available in other types of systems.

For example, changing a processing rule may result in another logic entity, such as configuration data, being inconsistent with underlying business logic. However, because business applications are large and complex, the user making the change may be unaware that the logic entity now contains a logic structure error. Traditional table-based business applications do not have a mechanism for identifying or optimizing inconsistencies in underlying business logic. Thus, many problems, such as broken or circular links in logic entities, may be undetected in the business application.

A number of well-known programming techniques may be used to optimize the design of existing software code without altering the code's behavior. For example, a software debugger, such as Validity Check, True Time, or Bounce Checker may improve code by testing it and modifying the values of variables where necessary. Optimized code is generally easier to read and understand and more efficient to maintain.

Methods and systems consistent with the present invention enable the use of known software tools, such as debuggers, to optimize the configuration of a business application by eliminating inconsistencies in underlying business logic of the business application. In one embodiment, each logic entity in a business application may be transformed into a corresponding object-oriented structure. An object-oriented language tool, such as a compiler, may then be used to check the structural consistency of the object-oriented structures and flag identified logic structure errors. Although the object-oriented language tool literally identifies problems, such as broken links, in object-oriented code, it also identifies problems in the underlying logic of a business application because the code models the business application. For example, an object-oriented compiler may identify a circular reference within a branch of code. By correlating that branch of code with its corresponding logic entity in the business application, the inconsistent logic entity may be identified. In one embodiment, inconsistent logic entities in the business application may be corrected after they have been identified using the object-oriented model, and the model translated back into business application entities that will function in the business application system.

For example, a debugging tool will optimize the object-oriented structures, for example, by "repairing" circular links. However, the operation of the code will not be affected. Once correction of identified logic structure errors in the object-oriented code is completed, the object-oriented code may be translated into corresponding business application logic structures. As a result, the business application will reflect the improvements made by optimizing the object-oriented code.

In one method consistent with the present invention shown in FIG. **13**, a business application composed in a first programming format may be analyzed by receiving a model representing the business application in a second programming format (stage **1310**) and processed by using a code optimization tool to identify logic structure errors (stage **1320**). Identified logic structure errors may then be corrected (stage **1330**) and applied in the first programming format, to the business application (stage **1340**).

Using CASE Tools to Verify Business Application

As described above, business applications often involve large and very complex table-based systems. To configure and implement a business application, a company may need to employ highly trained consultants. Due to the complexity of traditional business applications, it may take years to configure an application for a single company. Business applications typically have limited range of development tools to assist in development of the business applications configuration.

Computer-aided software engineering (CASE) tools assist a software developer in developing and maintaining software code. CASE tools enable users to design and manipulate code, such as object-oriented code, in an easy-to-understand manner. For example, data dictionaries and diagramming tools aid developers in analyzing and designing software systems. In another example, application generators may assist in actual programming of a system. Other CASE tools may assist with data modeling, reverse engineering, simulations, etc. Though CASE tools are available to assist programmers in analyzing and designing traditional software systems, such generic tools are not capable of being applied to a large, legacy system that might be customized to suit the needs of a single company.

Systems and methods consistent with the present invention enable the use of CASE tools to analyze and verify the configuration of a business application. By transforming business application structures into object-oriented structures, object-oriented CASE tools may be used to verify the underlying business application configuration. For example, CASE-tool-generated diagrams representing business application layers may assist an architect of a business application in visualizing a matrix of a project. As a result, the architect of a business application may more efficiently design and optimize a business process. For example, such diagrams may help a user to identify an amount of wagetypes, and how those wagetypes are related. In another example, simulation or modeling CASE tools may be used to test the effects of potential configuration changes in a business application. In this way, CASE tools could be applied to assist a consultant in configuring a business application or verifying an existing business application configuration.

Systems and methods consistent with the present invention enable business application consultants to generate and analyze the structure of business applications using tools from a CASE environment. For example, a developer may use a Modified Modeling Language (MML) tool to build a high level diagram of a logic entity, such as a business object class.

By transforming business application structures into object-oriented structures, object-oriented CASE tools may be applied to generate similar business application structures. For example, a user can write a new business rule using familiar object-oriented code, and that code may be converted into logic business entities to apply the new rule in the business application. As a result, users may configure the business application without worrying about its native format or structure.

In some embodiments, when business application structures are displayed using object-oriented code structures, the business meaning of the object-oriented code structures may also be displayed. This aids the user in understanding a com-

plex business application in the format of familiar, easy-to-understand object-oriented programming constructs.

FIG. **14** is an exemplary flowchart of a process for analyzing and verifying an application configuration using CASE tools. In certain embodiments consistent with the present invention, a business application composed in a first programming format is analyzed by receiving a model representing the business application in a second programming format (step **1410**). The first programming format may be, for example, a table-based format, and the second programming format may be, for example, an object oriented programming format. The model is then processed using CASE tools to generate its structure, including, for example, the structure of the business application configuration data and rules (step **1420**). In another example, processing may include generating a diagram of the model's object classes which represent logic entities within the business application. The generated structure of the model may be analyzed to depict a structure of the business application (step **1430**).

In certain embodiments, a table-based business application is represented by an object-oriented programming model. Because the model appears to contain object-oriented code statements, traditional CASE tools may be applied to it for a number of purposes, such as designing new structures (e.g., business application rules), testing proposed changes (e.g., a new tax category in the business application), and simulating a business application run (e.g., an end-of-month payroll run in the business application). Once these CASE tools have been applied to the object-oriented model, the results can be applied back to the business application to make appropriate modifications to the business application.

Comparing and Merging Business Applications

Large, complex legacy systems require almost constant updating and maintenance to keep up with the changing business environment. These changes may include, for example, software patches fixing technical glitches or improving the usability or the performance of a business application; new code to support changing business rules or evolving user needs, etc. For example, changing tax laws in one of the countries in a payroll business application may require changing many business rules within the application. Each change to a business application may affect many logic entities within the business application, possibly changing an outcome, for example, an amount of calculated payroll.

Traditional table-based business applications do not have any simple mechanism for modeling changes or identifying changes between different versions of a business application. A user attempting to compare two different versions of a business application and identify, for example, different tables or differences in a code, would have to compare each business rule separately, one by one. Because the business application is large and complex, this process may be extremely time-consuming and require a user with extensive training. Even then, small but important changes may be hard to recognize in a large table-based application.

Methods and systems consistent with the present invention allow a user to compare different versions of a business application using an object-oriented model of the business application. By transforming business application structures into object-oriented structures, such as object-oriented language constructs, individual differences between versions of the business applications may be identified and displayed to a user, for example, on a user interface.

To aid users in identifying differences between different versions of a business application, multiple versions of a business application configuration may be stored in a database, such as a configuration control database like CVS. Each version of a business application may be stored in a configuration control database as a transformed corresponding object-oriented structure.

Using methods and systems consistent with the present invention, to compare an updated version of the configuration of a business application with its previous versions, a user may extract the latest control version of an object-oriented model of the application from a configuration control database. Then, a user may extract the latest deployed version of a business application from a server database containing the deployed version of the business application, including recent updates, such as, for example, patches. The extracted deployed version may be transformed into a corresponding object-oriented structure. Finally, the user may extract a version of the business application including user-made current changes from the user's storage space. The object-oriented models of the business application may be capable of autonomous existence from the business application. Having all three versions represented in an object-oriented structure may allow a user to perform a three-way comparison between the object-oriented structure edited by the user and two retrieved versions copied from the configuration control database and the deployed server database.

Using methods and systems consistent with the present invention, a user may automatically compare all three versions of the business application model in a visually enabled mode by, for example, selecting a "Show Changes" button. As a result, business objects, for example, wage types, that differ between the three configurations may be displayed on an user interface marked with an icon to show where changes have been made. In one example, an icon may be shown to highlight each change and list values of the business object before and after the change.

Comparing the object-oriented structures of the model's three different versions, object-by-object, may allow a user to identify, for example, configuration data (e.g., wagetypes), business rules, schemas, and other logic entities where changes have been made. In each instance, a user may choose a value of the business object for preservation and subsequent storage as a new version of the business application model. Thus, as a result of the three-way comparison, a new version of the object-oriented structure may be created containing a user's chosen values for each business object.

When a user completes the three-way comparison and decision process, the newly-created version of the object-oriented model may be transformed into a newest version of the business application. That version may also be tagged as the "latest" and forwarded to a configuration control database for storage and/or to a target business system for deployment.

FIG. **15** is a flowchart of a process for comparing and merging versions of a business application using methods consistent with the present invention. A current version of a business application is determined by receiving a first version, a second version, and a third version of an object-oriented programming model representing the business application (stage **1510**). The objects of each of the first, the second, and the third versions of the object-oriented programming model are compared (stage **1520**) to identify differences between any objects in these models (stage **1530**). For each identified difference, a determination is made as to which version of the model is to be associated with the first version of the object-oriented model (stage **1540**). The first version of the object-oriented model is then assigned as the current version of the model (step **1550**). Using methods and systems consistent with the invention, the current version of the model may be translated back into the business application format, thus creating a current version of the business application.

Customer Support

As described above, OOP language tools, such as compilers, analyzers, optimizers, etc. can be used to find errors or problems in the configuration model of a business applications. Such errors or problems may include errors in relationships, calls, interactions, design, etc. associated with the modeled null functions.

FIG. **16** is a flowchart of an exemplary process consistent with the present invention. As shown in FIG. **16**, a model representing the business application composed in a second programming format is processed (stage **1610**). Systems consistent with the invention may then determine, based on the processed model, a potential problem in the business application (stage **1620**) and identify the determined potential problem by displaying a marker in the model of the business application (stage **1630**). In exemplary embodiments, the marker may be displayed in the model at a location where the potential problem occurs. Furthermore, the displayed marker may be associated with information describing the determined potential problem.

User Interfaces

FIGS. **17** to **56** illustrate exemplary user interfaces consistent with the invention for enabling a user to process a model representing a business application, as described above with respect to FIGS. **1** to **16**.

Consistent with methods and systems of the invention, a user interface displaying all the relevant information on one graphical and productive screen may be used. As shown in FIGS. **17** to **56**, user interfaces consistent with the invention may enable a user to easily navigate within an object-oriented structure from one object to another. The user interface may provide an instant visual alert of a potential problem, for example, with a source code. The user interface may also provide an intelligent code assistance and wizards. For example, all the references to a selected object, such as a schema or a rule, may be instantly displayed facilitating an instant dependency analysis. In another example, a user interface may have superior error detection and reporting capabilities, such as instantly reporting errors or enabling a user to find an exact error location just after one click.

Consistent with methods and systems of the invention, a user interface shown in FIG. **17** illustrates auto-complete capabilities in an Integrated Configuration Editor (ICE) environment. As shown in FIG. **17**, the user interface may list all the business objects and provides documentation for each of them. The user interfaces may depict grouping of the projects in ICE environment. The integrated configuration environment may include, for example, a series of windows displaying business application entities as a hierarchy of projects, rules, and schema. The environment may also include an object-oriented model of a business application object displayed together with documentation pulled from the business application.

For example, FIG. **18** depicts grouping of the projects defined by a country, and FIGS. **19** and **20** depict grouping of the projects defined by a system and a client, respectively, which has the least possible amount of attributes allowing to enforce uniqueness for all managed objects within a project. For example, using three attributes as the minimum set of attributes to define uniqueness, as shown in FIG. **19**, may allow the system to rearrange attributes and build hierarchies with better workspace organization. The user interface shown in FIG. **20** may depict a linear project layout when all key attributes are shown in a project name. A little block with a cross inside appearing next to a project type may indicate an existing potential problem or an error. For example, a block next to Germany on FIG. **18**, may indicate that an error in object "Germany" exists.

Consistent with methods and systems of the invention, a user interface shown in FIG. **21** may depict an offline debugger or log replay. Having an ability to see this screen may allow a user to look at dynamic and static views of payroll tables, full view of payroll steps, a source code displayed at a current execution point, and debugging controls.

Consistent with methods and systems of the invention, a user interface shown in FIG. **22** may depict inter object relationships within a project. For example, a user highlighting processing class with value M003, such as salary wage type, may result in displaying different rules referenced to that wage type.

Consistent with methods and systems of the invention, a user interface shown in FIG. **23** may depict one or more problems or errors encountered when business application structures are transformed into object-oriented structures. If during the conversion an error occurred, the error may be marked as a task for the user to fix. For example, if the user typed in an invalid syntax in a "RuleDPPF.java" module, a task of "Empty code" is listed for the user to fix along with a complete description of the error, which in this example is that "RuleDPP1 cannot be resolved."

Consistent with methods and systems of the invention, a user interface shown in FIG. **24** may depict, in greater detail, one or more problems, or errors, encountered when business application structures are transformed into object-oriented structures. If in FIG. **23** a user double clicks on a error, the schema where the problem exists is displayed for the user and the portion of the schema with the error is highlighted. When a cursor is placed above the marker, a pop-up box may show the details of the error. For example, the user may be notified that the method P0171 is undefined.

Consistent with methods and systems of the invention, a user interface shown in FIG. **25** may depict all the errors encountered in when business application structures are transformed into object-oriented structures. Each error may be indicated by a marker (here a rectangular flag) on the far right of the user interface identified by a code debugger applied to the object-oriented model. When a cursor is placed above the marker, a pop-up box may show the details of the error, and a click on the marker takes the user to the exact location of the error within the schema. For example, in the schema "RuleW020.java," errors are indicated by the marker on the right side of the screen. If a user places a cursor above the marker indicating the error, the details of the error "RuleWRFI cannot be resolved" is shown.

Consistent with methods and systems of the invention, a user interface shown in FIG. **26** may depict one or more warnings encountered when business application structures are transformed into object-oriented structures. A warning may consist of informing a user that a part of a module may be referencing a rule or value that is no longer used by the system. The warning is not an error and will not stop the conversion of the R3 structures to the Java structures. For example, if within a module a variant is checked to determine whether it equals a certain value, i.e., "processingclasses.ProcessingClasses04.Value8," the system may inform the user that "The field ProcessingClass04.Value8 is deprecated," in other words, ProcessingClass04.Value8 may not exist or is no longer used within the system.

Consistent with methods and systems of the invention, a user interface shown in FIG. **27** may also depict a problem, or error encountered when business application structures are

transformed into object-oriented structures. The error is marked as a task for the user to fix with a description being given under the "Problems" tab. For example, the error of "empty code" is listed as a task for the user in the module "RuleDPPF.java" and the description of the error is that "Rule9112 cannot be resolved." Rule9112 may also be highlighted in the user interface.

Consistent with methods and systems of the invention, a user interface shown in FIG. **28** may further illustrate details associated with a problem, or error encountered when business application structures are transformed into object-oriented structures where a rule in the transformation may not produce any result. For example, if the user clicks on the description of the error, the user may be directed to the portion of the "RuleDPPF.java" schema where the error occurred. In this example, a variable "variant" is set equal to nothing; therefore the user may have to enter a value to resolve the error.

Consistent with methods and systems of the invention, a user interface shown in FIG. **29** may also depict all the problems or errors when business application structures are transformed into object-oriented structures in one view. Each errors may be indicated by a marker on the far right of the user interface. When a cursor is placed above the marker, a pop-up box may show the details of the error. If a user clicks on the marker, the user is taken to the exact location of the error within the schema. For example, in the schema "RuleW020.java," there are problems indicated by the marker on the right side of the screen. If a user places a cursor above the marker indicating the problem, the details of the problem "RuleWRFI cannot be resolved" is shown.

Consistent with methods and systems of the invention, a user interface shown in FIG. **30** may depict the object-oriented structure that has encountered an error during the transformation of the business application structures into the object-oriented structures, and give a description of the error.

Consistent with methods and systems of the invention, a user interface shown in FIG. **31** may depict one or more evaluation classes within a project. A user may select an evaluation class business object, such as evaluation class 02, "Wage type assignment for pay statement," as shown in the upper center right pane with the tab labeled "Evaluation class." By processing the OOP object model of the business application using common OOP tools and methods, the system finds and displays all the business objects that are related to or reference the selected "Wage type assignment for pay statement" evaluation class 02 object, as shown in the bottom right hand pane with the tab labeled "References."

Consistent with methods and systems of the invention, a user interface shown in FIG. **32** may depict an Evaluation class within a project in an R/3 view.

Consistent with methods and systems of the invention, a user interface shown in FIG. **33** may depict the specifications for an evaluation class business object such as an Evaluations class within a project in an R/3 view. For example, a user interface may depict the specifications of Evaluations class "Wage type assignment for payroll account."

Consistent with methods and systems of the invention, a user interface shown in FIG. **34** may depict a screen where one or more operations are defined for one or more countries in an R/3 view. For example, the operation "ADDW Add wage type to subsequent wage type" may be defined for Canada, China, Denmark, Finland, France, Germany, and Great Britain. FIG. **35** is another user interface consistent with an embodiment of the invention.

Consistent with methods and systems of the invention, a user interface shown in FIG. **36** may depict one or more processing classes within a project, and the various operations that can be performed on each processing class is also depicted. For example, the processing class "Cumulation and storage at end of gross part" is shown in the user interface, and with a right click on that processing a menu is shown where a user may choose from one of many options such as "Copy Processing Class."

Consistent with methods and systems of the invention, a user interface shown in FIG. **37** may depict a Processing class within a project in an R/3 view.

Consistent with methods and systems of the invention, a user interface shown in FIG. **38** may depict the specifications for a Processing class within a project in an R/3 view. For example, a user interface may depict the specifications of Processing class "Cumulation and storage of time wage types."

Consistent with methods and systems of the invention, a user interface shown in FIG. **39** may depict an R/3 outline view of a rule representation along with the converted object-oriented code. For example, for the Payroll Rule "X015—Valuation of time wage types" the business application outline is on the left hand side of the screen and the middle of the screen shows the converted object-oriented code for the same rule.

Consistent with methods and systems of the invention, a user interface shown in FIG. **40** may depict an object-oriented outline view of a rule representation along with the object-oriented code. For example, the object-oriented outline of the rule "$401—Store excess amount from pre-tax 4-01K to after-tax 401-K" is shown on the left side with the converted object-oriented code for the same rule is in the middle of the screen.

Consistent with methods and systems of the invention, a user interface shown in FIG. **41** may depict a rule in a business application view. For example, the rule "X013 Es Grouping" is depicted as an R/3 business application.

Consistent with methods and systems of the invention, a user interface shown in FIG. **42** may depict a table instance of the rule in a business application view. For example, a table instance of the rule "U013" is depicted as an R/3 data structure.

Consistent with methods and systems of the invention, a user interface shown in FIG. **43** may depict an R/3 data structure schema, the object-oriented code of that schema, and the various references where the schema is used. For example, the schema "$APO—reoccurring benefits/Deductions and one time payments" is depicted along with the references that use the schema, which are "US00—Payroll Calculation" and "$US0—Payroll Calculation."

Consistent with methods and systems of the invention, a user interface shown in FIG. **44** may depict a function within an R/3 data structure schema, the implementation of the function, and other schemas where the function is used. For example, the function "P0377" may be shown along with the implementation of the function in the middle of the screen and the bottom right may show other schemas (ex: "BRB2," "BGE2," etc.) that use the function "P0337."

Consistent with methods and systems of the invention, a user interface shown in FIG. **45** may depict a table within a project and all the rules that use that table. For example, the table "Output table" is depicted and under "References," all the payroll rules that use the "Output table" are listed.

Consistent with methods and systems of the invention, a user interface shown in FIG. **46** may depict the object-oriented representation of R/3 data structure features shown in FIGS. **47-52**.

Consistent with methods and systems of the invention, a user interface shown in FIG. **53** may depict a wage type of a project. For example, the wage type "M003-Salary" may be displayed to the user. The references at the bottom of the interface may show others schemas that use this particular wage type within the same project.

Consistent with methods and systems of the invention, a user interface shown in FIG. **54** may depict another example of a wage type of a project. For example, the wage type "1055—Premium Pay" may be displayed to the user. The references at the bottom of the interface may show other schemas that use this particular wage type.

Consistent with methods and systems of the invention, a user interface shown in FIG. **55** may show an object-oriented version of the R/3 data stucture "Wage type group" as shown in FIG. **56**. Each wage type group may have one or more wage types associated with it. For example, the wage type group "0008 Basic Pay" may be associated with the wave types "SJ00 JS Hourly Wage, "2J20 JS Perfect perf bonus," etc.

CONCLUSION

Accordingly, as disclosed, systems and methods are provided for analyzing a business application's data, operations, and relationships and creating a corresponding object oriented programming (OOP) object model, comprising OOP objects, functions, and operators corresponding to the business application entities, and various uses involving the OOP model for improving the business application. The foregoing description of possible implementations consistent with the present invention does not represent a comprehensive list of all such implementations or all variations of the implementations described. The description of only some implementations should not be construed as an intent to exclude other implementations. One of ordinary skill in the art will understand how to implement the invention in the appended claims in may other ways, using equivalents and alternatives that do not depart from the scope of the following claims.

The systems and methods disclosed herein may be embodied in various forms including, for example, a data processor, such as a computer that also includes a database. Moreover, the above-noted features and other aspects and principles of the present invention may be implemented in various environments. Such environments and related applications may be specially constructed for performing the various processes and operations according to the invention or they may include a general-purpose computer or computing platform selectively activated or reconfigured by code to provide the necessary functionality. The processes disclosed herein are not inherently related to any particular computer or other apparatus, and may be implemented by a suitable combination of hardware, software, and/or firmware. For example, various general-purpose machines may be used with programs written in accordance with teachings of the invention, or it may be more convenient to construct a specialized apparatus or system to perform the required methods and techniques.

Systems and methods consistent with the present invention also include computer readable media that include program instruction or code for performing various computer-implemented operations based on the methods and processes of the invention. The media and program instructions may be those specially designed and constructed for the purposes of the invention, or they may be of the kind well known and available to those having skill in the computer/software arts. Examples of program instructions include, for example, machine code, such as produced by a compiler, and files containing a high level code that can be executed by the computer using an interpreter.

What is claimed is:

1. A method of determining a current version of a business application, comprising:

receiving a first version of an object-oriented model representing the business application, wherein the first version includes a plurality of objects;

receiving a second version of the object-oriented model representing the business application, wherein the second version includes a plurality of objects;

receiving a third version of the object-oriented model representing the business application, wherein the third version includes a plurality of objects;

comparing the objects of the first version, the second version, and the third version of the object-oriented model to identify differences between the first version, the second version, and the third version of the object-oriented model;

determining, for each identified difference, which version of the object-oriented model is to be associated with the first version of the object-oriented model; and

assigning the first version of the object-oriented model as a current version of the object-oriented model.

2. The method of claim **1**, further comprising:

generating the first version of the object-oriented model based on a configuration of the business application stored in a version control database.

3. The method of claim **1**, further comprising:

generating the second version of the object-oriented model based on a currently deployed configuration of the business application.

4. The method of claim **1**, further comprising:

generating the third version of the object-oriented model based on a modified configuration of the business application.

5. The method of claim **1**, wherein the business application is in a table-based format.

6. The method of claim **1**, further comprising:

transforming the current version of the object-oriented model into a current version of the business application.

7. The method of claim **6**, further comprising:

storing the current version of the business application in a version control database.

8. A method of comparing configurations of a business application, comprising:

generating a first object-oriented model representing a first configuration of the business application stored in a control database;

generating a second object-oriented model representing a second configuration of the business application that is currently deployed;

generating a third object-oriented model representing a third configuration of the business application that has been modified from the currently deployed configuration;

comparing objects of the first object-oriented model, the second object-oriented model, and the third object-oriented model to identify a difference between the first object-oriented model, the second object-oriented model, and the third object-oriented model;

analyzing the identified difference to determine which object-oriented model represents a desired configuration of the business application; and

designating the object-oriented model representing the desired configuration of the business application, based on the analysis of the identified difference.

9. The method of claim **8**, further comprising:

transforming the designated object-oriented model to create the desired configuration of the business application; and

replacing the configuration of the business application that is currently deployed with the desired configuration of the business application.

10. The method of claim **8**, further comprising:

transforming the designated object-oriented model to create the desired configuration of the business application; and

replacing the configuration of the business application stored in the control database with the desired configuration of the business application.

11. The method of claim **8**, further comprising:

displaying the identified difference together with a relevant portion of the first, second, and third object-oriented models using a single user interface.

12. The method of claim **8**, wherein the business application is in a table-based format.

13. The method of claim **8**, further comprising:

receiving a modification to the currently deployed configuration of the business application from a user via a user interface; and

enabling the user to request comparison of configurations of the business application by selecting a button on the user interface.

14. A system for comparing configurations of a business application, comprising:

a generating component configured to

generate a first object-oriented model representing a first configuration of the business application stored in a control database,

generate a second object-oriented model representing a second configuration of the business application that is currently deployed, and

generate a third object-oriented model representing a third configuration of the business application that has been modified from the currently deployed configuration;

a comparing component configured to compare objects of the first object-oriented model, the second object-oriented model, and the third object-oriented model to identify a difference between the first object-oriented model, the second object-oriented model, and the third object-oriented model;

an analyzing component configured to analyze the identified difference to determine which object-oriented model represents a desired configuration of the business application; and

a designating component configured to designate the object-oriented model representing the desired configuration of the business application, based on the analysis of the identified difference.

15. The system of claim **14**, further comprising:

a transforming component configured to transform the designated object-oriented model to create the desired configuration of the business application; and

a replacing component configured to replace the configuration of the business application that is currently deployed with the desired configuration of the business application.

16. The system of claim **14**, further comprising:

a transforming component configured to transform the designated object-oriented model to create the desired configuration of the business application; and

a replacing component configured to replace the configuration of the business application stored in the control database with the desired configuration of the business application.

17. The system of claim **14**, further comprising:

a displaying component configured to display the identified difference together with a relevant portion of the first, second, and third object-oriented models using a single user interface.

18. The system of claim **14**, wherein the business application is in a table-based format.

19. The system of claim **14**, further comprising:

a receiving component configured to receive a modification to the currently deployed configuration of the business application from a user via a user interface; and

a button on the user interface configured to enable the user to request comparison of configurations of the business application.

20. The system of claim **19**, wherein the button is displayed together with the identified difference.

* * * * *